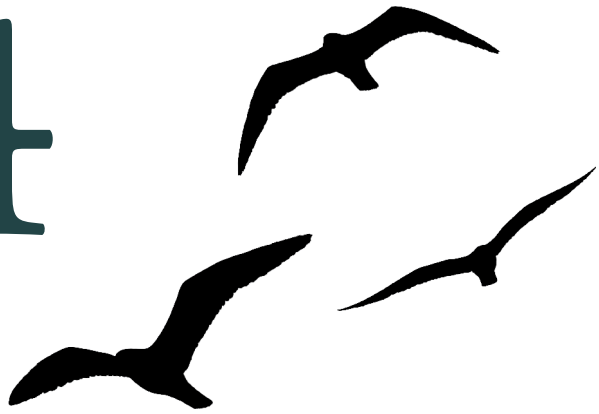
ICAPS 2014



Proceedings of the 2nd Workshop on

Planning and Robotics

Edited By:

Alberto Finzi and AndreA Orlandini.

Portsmouth, New Hampshire, USA - June 22, 2014

Organizing Committee

Alberto Finzi
DIETI-UNINA, Italy
AndreA Orlandini
CNR-ISTC, Italy

Program Committee

Rachid Alami, LAAS-CNRS, France
Amedeo Cesta, CNR-ISTC, Italy
Alberto Finzi, Università di Napoli Federico II, Italy
Robert Fitch, University of Sydney, Australia
Malik Ghallab, LAAS-CNRS, France
Joachim Hertzberg, University of Osnabrueck, Germany
Andreas Hofmann, MIT, USA
Felix Ingrand, LAAS-CNRS, France
Leslie Kaebling, MIT, USA
Sven Koenig, University of Southern California, USA
Jonas Kvarnström, Linköpings University, Sweden
Daniele Magazzeni, King's College, UK
Dinesh Manocha, University of North Carolina, USA
Maria Dolores Moreno, Universidad de Alcala, Spain
Karen Myers, SRI, USA
Daniele Nardi, Sapienza University, Italy
Andrea Orlandini, CNR-ISTC, Italy
Frederic Py, MBARI, USA
Alessandro Saffiotti, Orebro University, Sweden
Kanna Rajan, MBARI, USA
Reid Simmons, Carnegie Mellon University, USA
David Smith, NASA Ames, USA
Siddharth Srivastava, UC Berkeley, USA
Florent Teichteil-Königsbuch, ONERA, France
Felipe Trevizan, Carnegie Mellon University, USA
Manuela Veloso, Carnegie Mellon University, USA

Additional Reviewers:
Jonathan Cacace, Liron Cohen, Gustavo Goretkin, Andrea Micheli, Alessandro Umbrico, Tansel Uras.

PlanRob 2014 is partially supported by the SHERPA project (EU FP7 under the grant agreement ICT-600958). More information at http://www.sherpa-project.eu/

Foreword

Robotics is one of the most appealing and natural applicative area for the Planning and Scheduling (P&S) research activity, however, this potential interest seems not reflected in an equally important research production for the robotics community. On the other hand, the fast development of field and social robotics applications poses planning as a central issue in the robotic research with several real-world challenges for the planning community.

In this perspective, the goal of the PlanRob workshop is twofold. From one side, it aims at providing a fresh impulse for the ICAPS community to recast its interests towards robotics problems and applications. On the other side, its goal is to attract representatives from the robotics community to discuss their challenges related to planning for autonomous robots as well as their expectations from the P&S community.

The workshop aims at constituting a stable, long-term establishment of a forum on relevant topics concerned with the interactions between Robotics and P&S communities presenting a stimulating environment where researchers could discuss about the opportunities and challenges for P&S when applied to Robotics.

The first edition of the PlanRob workshop has gathered a very positive feedback and a major follow up was the organization of the ICAPS 2014 Robotics Track chaired by Felix Ingrand (LAAS-CNRS) and Leslie Kaelbling (MIT). This second edition of the PlanRob workshop has been proposed in synergy with this track to further enforce the original goal and to maintain a more informal forum where also more preliminary/visionary work can be discussed as well as more direct and open interactions/discussions may find the right place.

In our opinion, PlanRob'14 succeeded in achieving these objectives. Indeed, 20 papers have been accepted for oral presentation covering many relevant topics such as planning and execution for robots, high-level task planning, task and motion planning, multi-robot systems, goal reasoning and knowledge representation, etc. This seems to us a really good result for the workshop and, overall, it confirms a good feedback from the ICAPS community (but not only) on PlanRob topics.

Finally, two notable researchers have accepted our invitation to complete an already rich and interesting program: Brian Williams and Leslie Kaelbling from the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology (MIT).

Alberto Finzi and AndreA Orlandini
The PlanRob'14 Chairs

Table of Contents

Planning and Execution for Robots

Planning Under Temporal Uncertainty Using Hindsight Optimization 1
Scott Kiesel and Wheeler Ruml

A Flexible ANML Actor and Planner in Robotics 12
Fiip Dvorak, Arthur Bit-Monnot, Félix Ingrand and Malik Ghallab

HATP: An HTN Planner for Robotics 20
Raphaël Lallement, Lavindra de Silva and Rachid Alami

A Cooperative Model-based Control Agent for a Recongurable Manufacturing Plant 28
Stefano Borgo, Amedeo Cesta, Andrea Orlandini, Riccardo Rasconi, Marco Suriano and Alessandro Umbrico

Continual Planning via Reconfiguration and Goal Revision 38
Enrico Scala

Multi-Robot Planning

Multi-Robot Planning and Execution for an Exploration Mission: a Case Study 49
Guillaume Infantes, Charles Lesire and Cédric Pralet

Planning and Scheduling Single and Multi-Person Activities in Retirement Home Settings for a Group of Robots 59
Tiago Stegun Vaquero, Goldie Nejat and Chris Beck

Planning for Decentralized Control of Multiple Robots Under Uncertainty 69
Christopher Amato, George Konidaris, Gabriel Cruz, Christopher Maynor, Jonathan How and Leslie Kaelbling

Decentralized Adaptive Path Selection for Multi-Agent Conflict Minimization 80
Andrew Kimmel and Kostas Bekris

Efficient and Smooth RRT Motion Planning Using a Novel Extend Function for Wheeled Mobile Robots 90
Luigi Palmieri and Kai Oliver Arras

Knowledge Reasoning and Representation in Planning

Performance Level Profiles 99
Ronen Brafman, Guy Shani and Solomon Shimony

Iterative Goal Refinement for Robotics 106
Mark Roberts, Swaroop Vattam, Ronald Alford, Bryan Auslander, Justin Karneeb, Thomas Apker, Matthew Molineaux, Mark Wilson, James McMahon and David Aha

Challenges in Finding Ways to get the Job Done 117
Iman Awaad, Gerhard Kraetzschmar and Joachim Hertzberg

Integrating Declarative Programming and Probabilistic Graphical Models for Knowledge Representation and Reasoning in Robotics 127
Shiqi Zhang, Mohan Sridharan, Michael Gelfond and Jeremy Wyatt

Planning a Robot's Search for Multiple Residents in a Retirement Home Environment 136
Markus Schwenk, Tiago Stegun Vaquero, Goldie Nejat and Kai Oliver Arras

Task and Motion Planning

Synthesis of Plans for Robots 145
Srinivas Nedunuri, Sailesh Prabhu, Mark Moll, Swarat Chaudhuri and Lydia Kavraki

Heuristic Search for Task and Motion Planning 148
Caelan Garrett, Tomas Lozano-Perez and Leslie Kaelbling

Extending Knowledge-Level Contingent Planning for Robot Task Planning 157
Ron Petrick and Andre Gaschler

A Fast and Effective Online Algorithm for the Canadian Traveler Problem 166
Furkan Sahin, Vural Aksakalli and Ali Fuat Alkaya

Collision-free Path Planning for Remote Laser Welding 172
Andras Kovacs

Planning Under Temporal Uncertainty Using Hindsight Optimization

Scott Kiesel[1] and **Wheeler Ruml**[1,2]

[1]Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA

[2]LAAS-CNRS
7, avenue du Colonel Roche
31077 Toulouse FRANCE

Abstract

A robot task planner must be able to tolerate uncertainty in the durations of commanded actions and uncertainty in the time of occurrence of exogenous events. Sophisticated temporal reasoning techniques have been proposed to deal with such issues, although few existing planners support them. In this paper, we demonstrate the capabilities of a much simpler technique, hindsight optimization, in which uncertainty is handled by using sampling to generate deterministic planning problems that can be solved quickly. We find that sophisticated temporal reasoning is not required to handle many simple tasks. In comparison with a traditional temporal planner architecture, hindsight optimization is much simpler to implement while staying closely integrated with execution. It serves as a flexible baseline against which more complex methods can be compared.

Introduction

In many real-world domains, such as robotics, a planning agent does not have complete knowledge of the world state or precise control over action outcomes of actors and processes. Incomplete world knowledge, stochastic actions and exogenous events are obstacles a planning agent must tackle in many useful robotic applications.

Many planners make the assumption that an action, such as *pickup*, will have a deterministic outcome and duration. This is a fine assumption that makes planning much easier to reason about. However when the resulting plan is executed, actions can fail or take longer than anticipated.

Consider the domain of a robot office assistant. When issuing the simple task of picking up a set of keys from your desk, the planner will quite quickly emit the plan: *pickup(keys)*. When this plan is executed, the pickup action might fail. If the action fails, then the execution certainly will not result in a goal state. The ability to reason about action outcome uncertainty becomes very important when a plan will be executed.

Perhaps the goal state is slightly more interesting and the agent should pickup your keys and give them to you when you leave to go home. The other half of the assumption about actions that many planners make is that their duration is a known constant. However, depending on the starting pose of the robot office assistant or the position of the keys, this simple *pickup* can have a varying range of execution durations. If it could take anywhere between 1 minute and 10 minutes for your keys to be picked up, you might appreciate a planner that can take this range into consideration, instead of causing you to be 10 minutes late going home. If a planner only assumes the best case for action durations, it is easy to see that the agent could be late to a rendezvous. On the other hand, if the planner only assumes the worst case for action durations, the agent may spend time waiting at a rendezvous point unnecessarily. It could instead be performing more productive tasks with this time. The ability to handle action duration uncertainty becomes very important if a productive and punctual agent is desired.

An even more realistic situation for a robot office assistant to face is the task of retrieving your keys from a set of possible locations. It is not uncommon to forget exactly where you left your keys. Many planners however, require that the exact location of the keys be known when planning begins. If you need to manually search out your keys to simply set the initial state for your planner, you might as well skip using your planner because you have already found your keys. It is important for a planner to be able to handle location uncertainty if the exact state of the world is not known.

As hinted at in the previous example scenarios, the agent may not be the only entity in its world. Other agents may exist, such as yourself, and interaction with these other agents is only natural. These other agents are not necessarily under the control of the same planner as your robot office assistant. The world can be affected and changed outside of the control of the planner. Maybe you found your keys while trying to fully annotate an initial state for your planner and finding your keys is no longer a goal for the agent. Let's consider a new goal, before you leave the office you would like your robot office assistant to give you a coffee for the trip home. The coffee is not essential for you to get home, but it makes the trip significantly more pleasant. As such, you are willing to wait for 10 minutes before you depart without your coffee. As a human, your timing is not always exact so you might leave sometime between 5pm and 5:30pm. If you would like to get your coffee frequently before you leave the office it is important for a planner to be able to reason about the temporal uncertainty associated with interactions involving other agents and events exogenous to the planner.

Adding a single one of these three aspects of uncertainty

to a domain renders many planners inapplicable. Adding all three types of uncertainty further reduces the number of applicable planners. Those algorithms previously proposed to handle these uncertainties are complicated and can rely complex data-structures such as a Simple Temporal Network With Uncertainty (STNU) (Morris, Muscettola, and Vidal 2001). Many of them also require reasoning about all of the unknown factors at once requiring complex world representations.

In this paper we introduce the Temporal-Uncertainty Hindsight Optimization Planner (TU-HOP), a simple and straightforward approach that extends previous work on hindsight optimization. Instead of trying to manage the various uncertainties directly, we employ a basic sampling strategy and a deterministic planner. TU-HOP begins with a set of beliefs about the initial world state. This belief state manages certain and uncertain information about locations, arrivals and departures of objects and agents and expected action outcomes and durations. Given the current belief, a set of deterministic world samples consistent with the belief state are generated and then solved by the deterministic planner. Using the resulting solutions, the next action is chosen based on solutions maximizing overall expected reward. This action is executed, the belief about the world is updated based on the action's result and the process starts again.

We show that TU-HOP is simple to understand and implement. We also show that TU-HOP is very capable of solving problems containing uncertain action outcomes and durations, uncertain object and agent locations as well as exogenous events. Its simplicity and capability make it a flexible baseline against which future temporal uncertainty planning research can be compared.

Previous Work

There is wealth of literature on deterministic domain dependent and independent planners. We are able to leverage this previous work, as others have, to incorporate well researched deterministic planning ideas and concepts into a larger framework (Yoon, Fern, and Givan 2007; Shani and Brafman 2011).

Yoon, Fern, and Givan (2007) incorporate a classical domain independent planner called FF (Hoffmann and Nebel 2001) into their planning framework called FF-Replan to solve problems with uncertain action effects. FF-Replan uses FF to find a plan to carefully constructed deterministic version of the problem. It then executes actions according to the plan until the executed action has an unexpected effect or the goal is achieved. If an unexpected effect is observed before achieving the goal, FF is called once again to construct a new plan from the current state.

Temporal Planning and Execution

EUROPA (Barreiro et al. 2012) is class library and tool set for building planners within a temporal planning paradigm. It is a complicated architecture that handles time and resources and constructs plans offline. It is able to handle many temporal events using its modeling language NDDL. It relies on many handcoded internal modules and does not directly handle the uncertainties of object locations or action outcomes.

IxTeT (Ghallab and Laruelle 1994; Laborie and Ghallab 1995) is an complex offline planning and scheduling system that can handle time and resources by constructing partial order plans and resolving *threats* to the achievement of goals during planning. It strives to find a balance between the planning (what to do) and the scheduling (in what order to do it) addressing many domains in the intermediate spectrum between planning and scheduling. It however does not take into account the uncertainties that become evident during execution of plans.

Procedural Reasoning System (PRS) (Ingrand et al. 1996) is a system for supervision and control of autonomous mobile robots. This system is explicitly able to monitor plan execution and provide feedback on action execution to an underlying planning system. However, PRS still relies on a high level planner to provide the high level actions for it to execute. Integration of TU-HOP with PRS is a promising avenue for future research.

IxTeT-EXEC (Lemai and Ingrand 2003) is complex system that allows for execution control, plan repair and replanning. IxTeT-EXEC is an extension of the IxTeT planner integrated with PRS (and several other layers). IxTeT-EXEC is able to handle temporal constraints (inherited from IxTeT) as well as action failures and unpredicted action outcomes as reported by PRS. However, this is a very complicated system that is non-trivial to implement and does not address the aspects of temporal uncertainty of interest in this paper.

Simple Temporal Network with Uncertainty (STNU) (Morris, Muscettola, and Vidal 2001) are an extension of Simple Temporal Networks (STN) (Dechter, Meiri, and Pearl 1991). In many cases of interest in planning, an STNU would be used to determine dynamic controllability. If an STNU is dynamically controllable, then we can be assured that from the current state, the actions we plan to execute will not cause us to violate any future temporal constraints regardless of their outcome durations. Computing dynamic controllability, is $O(n^3)$ in the general case, where n is the number of edges in the network (Nilsson, Kvarnstrom, and Doherty 2014).

Dearden et al. (2003) outline an approach for constructing contingent plans incrementally that handle uncertainty in action duration and resources. While this approach is very capable, it is signficantly more complicated to implement than TU-HOP.

Prottle (Little, Aberdeen, and Thiébaux 2005) is a probabilistic temporal planner that deals with uncertain action outcomes, action durations and exogenous events. Prottle handles action durations that are specified as a finite set of possible values. The underlying planner framework relies on the ability to enumerate all possible successors from a given state. When action durations can take on any value in a continuous range, it becomes infeasible to enumerate all of the duration possibilities.

HYBPLAN (Mausam 2007) is a hybrid MDP solver built from GPT (Bonet and Geffner 2001) and MBP (Bertoli et al. 2001). While HYBPLAN can handle concurrent and durative actions it does require the implementation of two other

planners making it much more difficult to implement than TU-HOP.

Hindsight Optimization

Hindsight Optimization was originally developed for scheduling and networking problems (Chong, Givan, and Chang 2000; Mercier and van Hentenryck 2007; Wu, Chong, and Givan 2002) and has been used in a probabilistic planning setting (Yoon et al. 2008; 2010). In these previous applications, sampling was used to resolve uncertainty in the outcome of actions. Burns et al. (2012) used hindsight optimization to solve a problem where exogenous goals are arriving, which requires the agent to plan ahead and anticipate these arrival events. Most recently, hindsight optimization was applied to open world planning where sampling was used to resolve uncertainty in object existence and location as well as uncertainty in navigation graph connectivity (Kiesel et al. 2013).

Similar to most sampling techniques, the samples of generated possible worlds used in this paper are intentionally not exhaustive. They are intended to provide useful relative judgments on the expected value of actions. In hindsight optimization, given a current world state, we are faced with the choice between all applicable actions. The first step in hindsight optimization is to generate possible world samples that could be true according to the current world belief state. Then in order to estimate the value of an action, we apply that action in each of the sampled possible worlds, find deterministic plans from each of the resulting states, and average over the resulting reward yielded by each plan. The action with the highest average plan reward over the sampled worlds is chosen to be executed.

More formally, we define the value of being in a state s_1 as the maximum expected reward over plans that extend from s_1. That is, the maximum reward over all possible future action sequences of the total reward over all possible future states:

$$V^*(s_1) = \max_{A=\langle a_1,\ldots,a_{|A|}\rangle} \; \underset{\langle s_2,\ldots,s_{|A|}\rangle}{E} \left[\sum_{i=1}^{|A|} R(s_i, a_i) \right]$$

where $R(s, a)$ represents the reward of performing action a in state s. Given our expectations about the uncertainties in the world, we would like to find the action sequence $A = \langle a_1, ..., a_{|A|} \rangle$ that maximizes the expected sum of action rewards. To compute V^* exactly, we would need to compute the expectation for each of infinitely many plans.

In hindsight optimization, we approximate the value function by exchanging expectation and maximization, so that we are taking the expected value of maximum-reward plans instead of the maximum over expected-reward plans:

$$\hat{V}(s_1) = \underset{\langle s_2,s_3,\ldots\rangle}{E} \left[\max_{A=\langle a_1,\ldots,a_{|A|}\rangle} \sum_{i=1}^{|A|} R(s_i, a_i) \right]$$

The expectation in this approximation of $V^*(s)$ can be computed using sampling and fixed values for each of the uncertainties in each maximization. As in other applications of hindsight optimization, the stochastic elements have been reduced to known values by sampling. For each possible value that an uncertainty could take on in the expectation, the problem is to maximize reward given a known world, i.e., standard, deterministic, reward-maximizing planning. As the underlying deterministic problem becomes more difficult to solve with a standard deterministic planner, TU-HOP can employ a limited horizon planner. A limited horizon planner uses a time horizon which is simply a temporal value by which search depth is bounded. Setting this bound to infinity results an informed, full solution to the maximization problem, decreasing the horizon results in greedier behavior, only considering more immediate reward.

TU-HOP(W, N, H)
1. while true
2. for i from 1 to N do
3. $w_i \leftarrow sample\_world(W)$
4. foreach action a
5. $s' \leftarrow a(s)$ /* sampled outcome of $a(s)$ */
6. $r \leftarrow (\sum_{i=1} solve(s', w_i, H))/N$
7. $Q(s, a) \leftarrow R(s, a) + r$
8. $a_{best} \leftarrow argmax_a \, Q(s, a)$
9. res = execute(a_{best})
10. update(W, a_{best}, res)

Figure 1: The TU-HOP planner.

We define the Q-value to be the cumulative expected reward of taking an action a_1 in state s_1:

$$Q(s_1, a_1) = R(s_1, a_1) + \underset{\langle s_2,s_3,\ldots\rangle}{E} \left[\max_{A=\langle a_2,\ldots,a_{|A|+1}\rangle} \sum_{i=2}^{|A|+1} R(s_i, a_i) \right]$$

From this, we estimate the best action choice in s as $\max_a Q(s, a)$. Using this technique, we are said to be performing optimization with the benefit of 'hindsight' knowledge about how future uncertainty will be resolved. It is important to point out that this action choice strategy is an unsound reasoning technique (Yoon et al. 2010).

After the best action is executed, hindsight optimization updates its current belief state based on the outcome of its action and how it has affected the world. This newly updated belief state will be used in the next planning step.

Approach

In this paper, we extend the line of work on hindsight optimization to handle the three types of uncertainties previously discussed; uncertain action outcomes and durations, uncertain object and agent locations and exogenous events. TU-HOP is an online planner that interleaves search and execution, emitting single actions for the controllable agents to execute at a time.

The pseudo-code in figure 1 provides a high level summary of the TU-HOP planner. The planner first receives

three parameters, the first is the current belief about the world state, the second is the number of samples to be used and the third is the horizon with which to bound the deterministic solver. First, we generate a set of N possible worlds that are consistent with the planner's current belief about the world (lines 2–3). Each of these sampled worlds are deterministic versions of the current belief where all objects have known locations, actions have known outcomes and events have known start and end times. Next, for each action a in the domain, we consider the resulting state $s' = a(s)$ (line 5). Then, each possible world w_i is initialized with the state s', generating a fully-known deterministic planning problem. Solving this problem provides an estimate of the reward from s'. The mean reward across the set of samples (line 6) along with the reward of the action $R(s, a)$ is used as the Q-value for each action a in the original state s (line 7). We then select the action with the maximum Q-value (line 8), this action is then executed (line 9). The result of this action, success, failure or an inaccurate belief is returned and the current belief of the world state is updated with this information (line 10). With this new belief about the state of the world, the planner returns to the beginning of the loop and executes another iteration.

Robot Office Assistant Domain

In this paper we focus on a specific domain where a set of controllable and non-controllable agents are able to navigate around a topological map containing objects. Controllable agents are able to *pickup*, *putdown*, and *give* objects. The *give* action is the exchange of one object in an agent's possession to another agent. All of the information about the domain and belief of the world state will now be discussed.

The first major entity in the world belief is a representation of the topological navigation graph. Each node in the graph is connected to at least one other node in the graph via an edge. Each edge has a traversal success rate, a minimum and maximum successful traversal time and a minimum and maximum failure time. The success rate represents the probability that traversing this edge will succeed. The minimum and maximum successful traversal times provide an expected interval of how long it will take to traverse the edge if the traversal is successful. Similarly, the minimum and maximum failure interval describes how long it will take for the attempted traversal to report failure.

The objects present in the world each have two associated temporal intervals and a set of possible node locations. The first interval is the minimum and maximum expected arrival time for the object. This can be used to represent an object being dropped off by an agent outside the control of the planner. The second interval is the minimum and maximum duration the object is expected to remain usable in the world. This can be used to impose a deadline on when an object may need to arrive at its goal destination. A single node location represents complete certainty of the object's starting location. A set whose size is greater than one represents uncertainty of the object's starting location.

Agents are very similar to objects with two major differences. The first is that an agent can be marked as outside of the control of the planner. This is useful if an agent is only "stopping by" to receive an object from another agent that *is* under the planner's control. The second difference is that each agent also has a number of grippers that are available to hold objects.

Goals can be one of three types different types. The first is a simple *goto* goal which tells the planner which agent needs to be moved to which location. The second type of goal is *move*, which tells the planner which object needs to be moved to which location. The last type of goal is *give* and tells the planner which object should be given to which agent. Each specified goal also has an associated reward that will be received for achieving that goal. This allows for goal preferences in planning problem instances. It should also be noted that not all agents and objects need be involved with a goal.

Lastly, there are four actions in the domain; *pickup*, *putdown*, *give* and *no-op*. Please keep in mind that the implicit *move* action is defined on an edge by edge basis in the graph. The motivation behind this is that some edges may be more difficult or simply take more time to traverse. Each action has a success rate, a minimum and maximum successful execution duration and a minimum and maximum failure duration with the exception of the *no-op* action. The success rate represents the probability that executing an action will succeed. The minimum and maximum successful execution times provide an expected interval of how long it will take to complete the action if the execution is successful. Similarly, the minimum and maximum failure interval describes how long it will take for the attempted execution to report failure. The *no-op* action is always successful and has a deterministic execution duration equal to the planned duration. The duration for each *no-op* is determined during planning. It is set to be the time from the current state, until the next occurring event. An event is simply the arrival or departure of an agent or object, or another agent completing its current action. Setting the duration in such a manner can render a deterministic planner incomplete in certain domains, but in this simple Robot Office Assistant Domain completeness is maintained when planning in the deterministic world samples.

A Closer Look & Implementation Details

All of the domain instance information is managed and updated in the belief state of the TU-HOP planner throughout its execution. The planner is only able to observe and update facts in its belief state if an agent under its control is able to observe information about that fact. Agents are only able to observe facts about events, objects and other agents colocated with their current location. The planner observes the world through its controlled agents' actions. For example, if there is uncertainty about an object's location and an agent tries to pick up an object in an incorrect location, the agent is able to observe that the action failed because the object is not present in this location.

Before emitting any action to be executed, TU-HOP enters its first planning iteration. It begins by generating a set of possible deterministic world samples consistent with the current belief state. This means that in each sampled world any and all uncertainty is removed. This is achieved for the

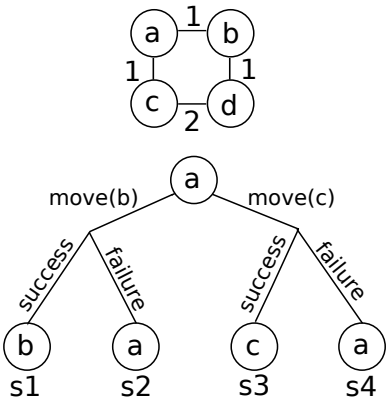


Figure 2: A simple example of the first step of hindsight optimization.

location uncertainty by picking, at random, one of the locations in each possible location list for the agents and objects. The action outcome uncertainty is resolved by using the success rate, success interval and failure interval for each action and constructing a deterministic mapping of times to outcomes and durations. This is accomplished by lazily querying the action in the deterministic world sample when needed for its outcome and duration given the current time. If the time has no mapping, the outcome is randomly computed given the success and failure values, then stored in a lookup table. If the time has a mapping already, that mapping is returned. In our implementation these time values are rounded to a hundredth of a second before doing the lookup. The arrival and departure times of any object or agent are also resolved by taking a random sample from the arrival interval and the duration interval to construct an exact arrival and departure time.

Following the hindsight optimization framework, in each world sample, TU-HOP examines each available action from the current state. In the most simple *goto* (navigation) case with no objects, TU-HOP will evaluate what will result after moving to each node adjacent to its current node. In figure 2 the agent is currently in location (a) and is considering moving to location (b) or (c). Each move action can either succeed or fail as depicted. Each of these outcomes, $\{s_1, s_2, s_3, s_4\}$, is generated and then reward is maximized individually in each outcome. In our implementation we use a very simple temporal horizon bounded breadth first search to evaluate reward in each outcome. The reward for execution of the action in a single deterministic sample is then a weighted mean of the reward achieved in the success and failure case weighted by the likelihood of the action succeeding and the likelihood of the action failing. For example, if the achievable reward under s_1 is 1, the achievable reward under s_2 is 0.5, and the success rate for that action is 0.9 (0.1 failure rate), then the reward achievable for executing *move(b)* is 0.95 by TU-HOP's reckoning.

This procedure is executed for each generated deterministic sample. Then the reward is averaged over all the samples yielding an estimate of achievable reward. The action with the highest expected reward across all samples is then selected for execution.

The action selected is then executed and the result of the action is used to update the belief state of the planner. This would include increasing the current time, removing a location from an object's possible location list, decreasing the size of an agent's arrival interval, and so on.

Experimental Results

We now evaluate TU-HOP by stressing each of the three types of uncertainties (location, action outcome and temporal uncertainty). All experiments were planned and executed in simulation on a Lenovo W520 with an Intel Core i7 and 8GB of RAM. For each instance problem considered, a set of 25 seeds were used when generating a grounded simulation world as well during planning in each world sample constructed from the current belief. All plots presented show a line representing the mean y-value across the instances and vertical lines representing the 95% confidence interval at that point on the line.

Location Uncertainty

The first set of experiments begin by issuing the goal of moving an object from its start location to a goal location. The easiest instance starts with the object's location known exactly. We increase the difficulty of the instances by adding uncertainty about the objects location to possibly two locations, then possibly three locations and so on. As the uncertainty about the object's start location increases, the agent should be forced to search out the true location. The results from this experiment using planner configurations of a horizon of 90 seconds and 1, 5 and 10 samples are shown in figure 3.

In all instances, from where the object is in a known location and up to 5 possible locations, the planner is able to find the object and deliver it to its destination. In figure 3 (b) and (c) the lines representing 5 and 10 samples show that these configurations required fewer actions to achieve the goal than the single sample configuration and also have an overall shorter goal achievement time. Since the planner was able to achieve the goal in all instances, we can see that by increasing the number of samples taken, the agent will visit the possible locations in a more reasonable order. First visiting the closest possible location, then moving to the next closest, and the next, until the object's true location is found. Figure 3 (a) shows the time required by the planner at each iteration. Even on the hardest instance with 10 samples the planner takes much less than a second on average before emitting an action for execution.

We also ran a small experiment to show the underlying planner's ability to scale with the number of object relocation goals. We start with a single goal of moving one object to a location and slowly increase the number of goals and objects in the world. The results from this experiment using planner configurations of a horizon of 90 seconds and 1, 5 and 10 samples are shown in figure 4.

In all instances the planner was able to move all the objects to their goal locations. We can see that as expected

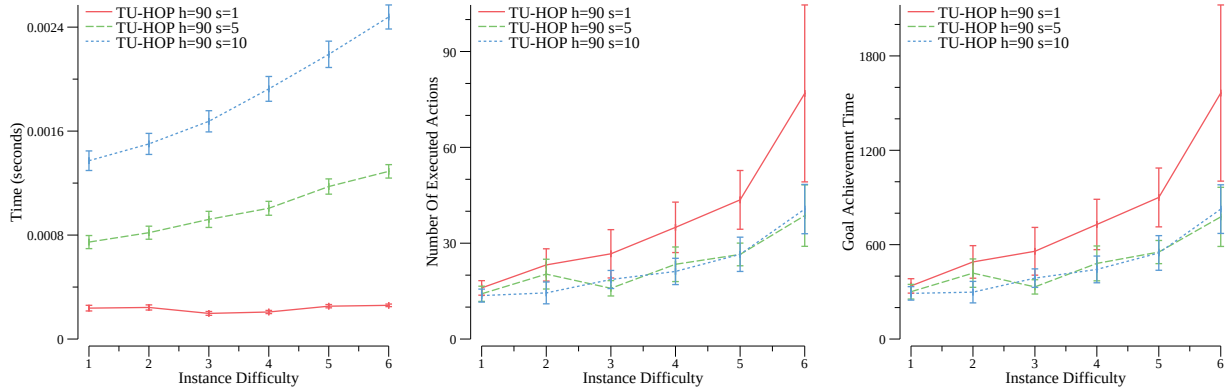


Figure 3: Increasing the amount of uncertainty in an object's location in the world between 1 and 5 locations using a horizon of 90 seconds and 1, 5 and 10 deterministic samples.

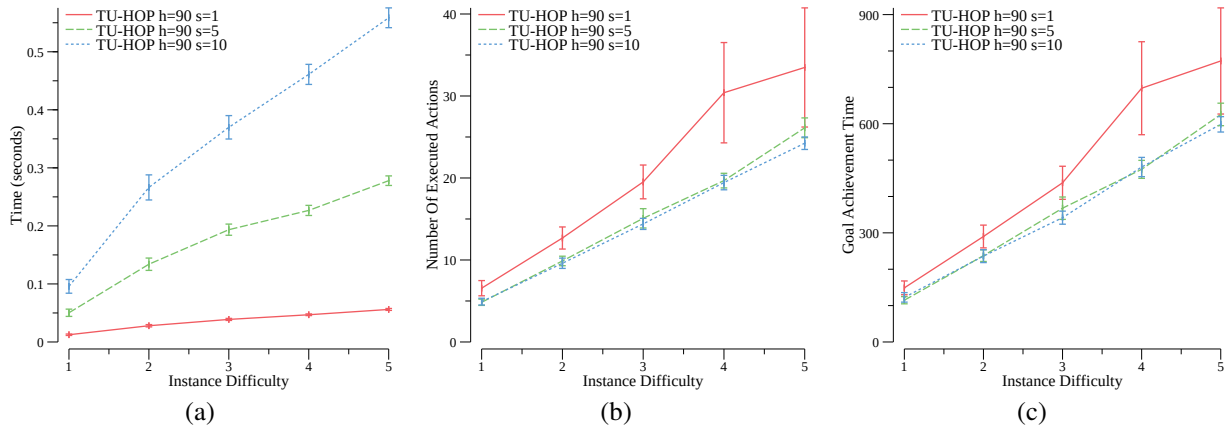


Figure 4: Scaling the number of objects in the world between 1 and 5 using a horizon of 90 seconds and 1, 5 and 10 deterministic samples.

in figure 4 (a), using more samples does increase the overall planning time at each planning step. However, even in the hardest considered instance with 5 objects using 10 samples the planner only took on average 0.6 seconds before returning the next action. The scaling could be significantly improved by using a heuristic or a more informed search technique than breadth first search. In figure 4 (b) and (c) a positive trend is shown where using more samples results in fewer actions in the final execution and also earlier goal achievement times.

Action Outcome Uncertainty

In the second set of experiments we issue a similar goal of moving an object from its start location to a goal location. However in these instances the object's location is known exactly and the topological edge traversals required complete achieve the goal will have increasing traversal failure rates. We start with a failure rate of 0% and increase it all the way to 50%. By increasing the edge traversal failure rates the agent will be forced to re-plan and accommodate for the failure. The results from this experiment using planner configurations of a horizon of 90 seconds and 1, 5 and 10 samples are shown in figure 5.

The planner was able to achieve all goals in all instances during this experiment. The x-axis in figure 5 is numbered by instance difficulty where a difficulty of 1 represents fully reliable edges. This means the outcome of traversing them has 100% probability of success. As the number increases, the success probability decreases to $\{90\%, 80\%, 70\%, 60\%, 50\%\}$. In figure 5 (a) we can see that the planning times for the 1, 5 and 10 sample cases are all quite similar until the edges become unreliable (50% success rate). As the failure rate increases, the plan lengths will increase and planning with more samples magnifies this in its overall planning time. In figure 5 (b) and (c) we see a trend similar to the last experiment. This is most likely caused by the same issue. The noise between sampled worlds is minimized by generating more samples.

A simple third set of experiments extending the second set were also performed. The instance is created with a set of inexpensive topological edge traversals between the agent and the object's start locations with high action failure rates. A secondary set of expensive edge traversals with very low failure rates are also created. As the inexpensive route becomes more unreliable throughout the experiments, the agent should choose to take the more reliable expensive route. The results from this experiment are shown in figure 6.

Again, in this experiment the planner was able to achieve all goals in all instances during his experiment. Figure 6 (a) shows a predictable trend where increasing the number of samples causes the planning step between action executions to increase. However, planning times with 10 samples on the most difficult instance are still well below 0.1 seconds on average. Figure 6 (b) and (c) show the realization of our prediction. When using only a single sample the number of actions in the final execution is lower for the first three instances, but the overall goal achievement time for those instances is higher than when using 5 and 10 samples. Once the reliability of the cheap edges decreases significantly in the last three instances, the single sample case starts to have longer execution lengths and continues to have later goal achievement times than the 5 and 10 sample cases.

Temporal Uncertainty

In this fourth set of experiments we involve a second agent outside of the control of the planner. The goal issued in this set of experiments is to give an object to this second agent and also relocate a second object. Adding a second object goal forces the planner to choose an ordering for the two goals which becomes very important in this experiment. The second agent does not begin at any location but is scheduled to arrive at one during a predefined interval and will only remain for a duration between some minimum and maximum value. We begin by starting with a small arrival interval and a long duration for the second agent. This makes it very easy for the planner to achieve both goals. We increase the difficultly of these instances by making the arrival interval larger (more uncertain) and decreasing the duration the agent remains before departing. By increasing the uncertainty, it becomes much more important for the planner to consider the possible arrival and departure times of the second agent if it is to catch it before it departs. The results from this experiment using planner configurations of a horizon of 90 seconds and 1, 5 and 10 samples are shown in figure 7.

In this set of experiments using TU-HOP with 1 and 5 samples was not able to achieve all goals in all instances. When the planner failed to achieve all goals, no point is included for the configuration in Figure 7 (a), (b) or (c). Panel (a) shows that planning times between action executions are still less than 1 second for almost all instance difficulties which is certainly acceptable when the action executions times for a robot can be in the range of full seconds to a minute for some actions. In figure 7 (d) the decreasing reliability of both goals being achieved is illustrated as the instance becomes more difficult. At first with the larger delivery window, the 1, 5 and 10 sample cases are able to achieve both goals reliably. However when the window is reduced the planner clearly benefits from more samples as the 10 sample case is the only configuraiton able to achieve all goals in all instances. Figure 7 (b) shows that as the delivery window constraints become tighter the agent will use more actions to attempt to give the object to the second agent. In panel (a) and (c) we can see the impact of the hardest instance on the overall planning time, and therefore the goal achievement time.

Discussion

The main two attractive features of hindsight optimization are its simplicity and generality. There are no obvious impediments to combining the current work with previous efforts that use hindsight optimization to address other forms of uncertainty such as stochastic action effects, arrival of additional goals, partial observability, or open worlds (Yoon et al. 2008; 2010; Burns et al. 2012; Kiesel et al. 2013).

Compared to a planner using an STNU, TU-HOP's handling of intervals is imprecise, thus the combinations of cir-

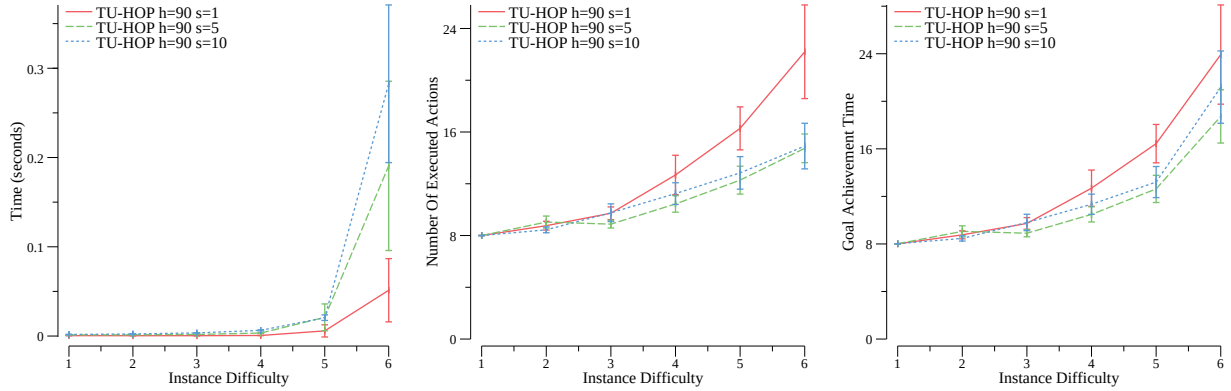


Figure 5: Decreasing the reliability of the only edges available to achieve an issued goal using a horizon of 90 seconds and 1, 5 and 10 deterministic samples.

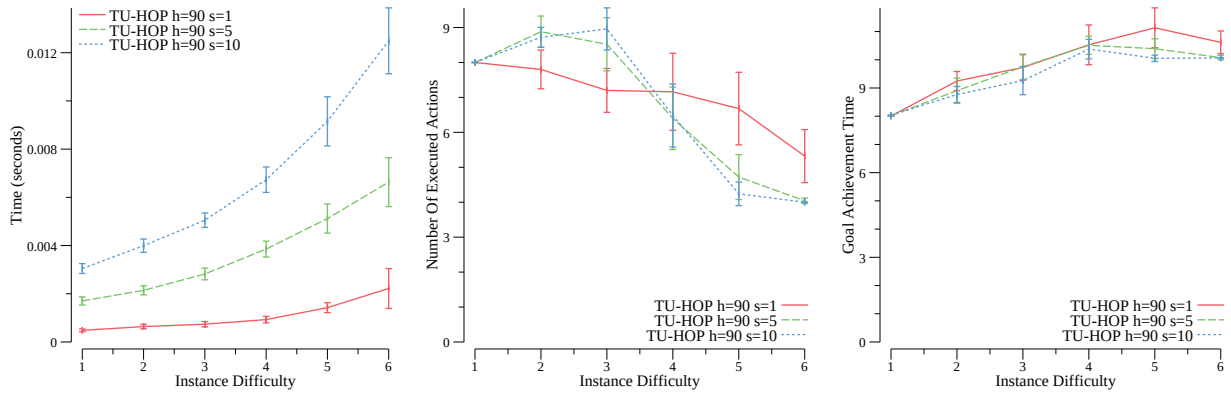


Figure 6: Decreasing the reliability of low cost edges forcing more reliable expensive edges to be utilized using a horizon of 90 seconds and 1, 5 and 10 deterministic samples.

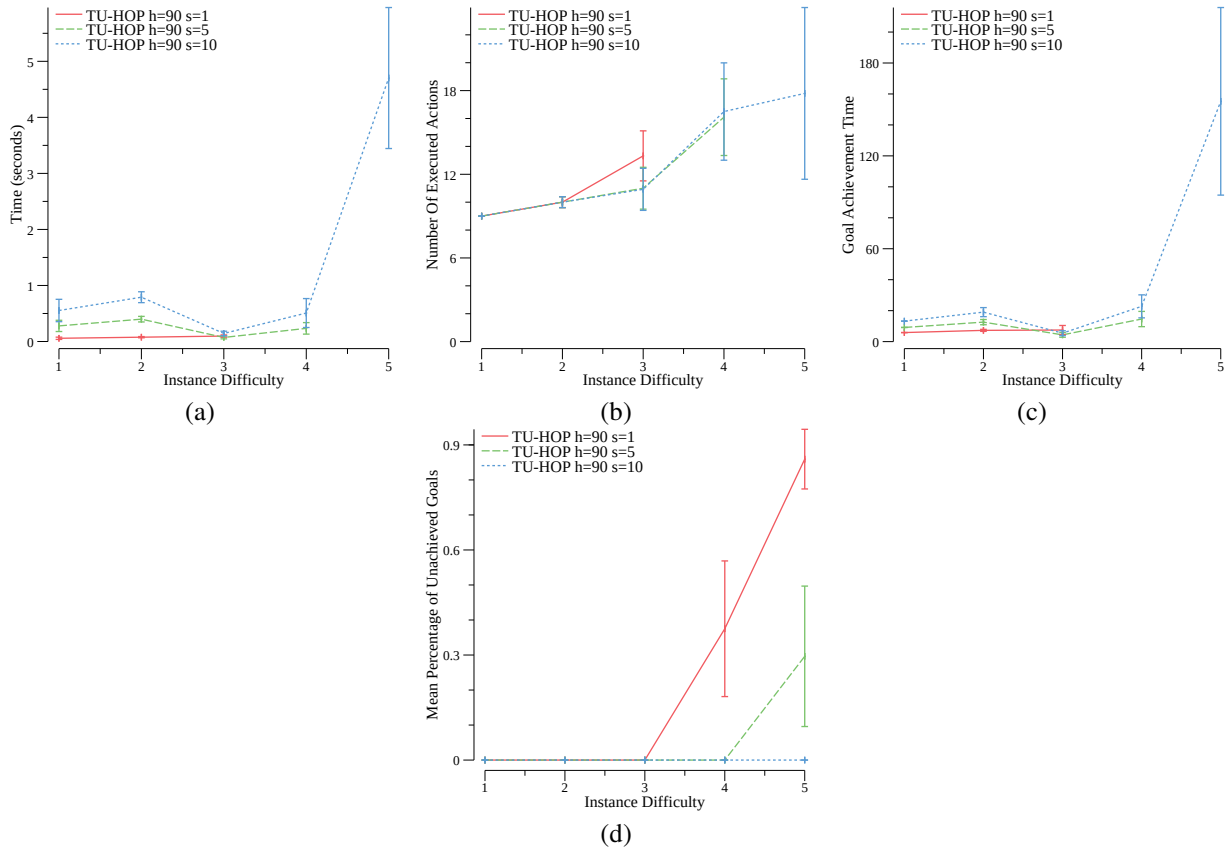


Figure 7: Increasing the uncertainty about when an agent will arrive while also decreasing the duration the agent will wait after arriving using a horizon of 90 seconds and 1, 5 and 10 deterministic samples.

cumstances that are anticipated is incomplete. This limitation gets more serious as the number of combinations of stochastic events that need to be considered increases. However, in many applications, it is not necessary to reason about such long chains of events in order to act successfully.

TU-HOP demonstrates one way of very tightly coupling planning and acting, namely to ensure reactivity by planning after every state transition and never explicitly committing to actions beyond the one that is currently executing.

Unlike many other task planners, TU-HOP does explicitly consider action failure when selecting actions. In this examination, a simple action model is used where actions only have two outcomes, success or failure, with varying execution durations. If actions were to have a more complicated, larger set of possible outcomes, more samples might be required to get a better approximation of how the actions will behave.

Action durations in this work were outside of the control of the TU-HOP planner, with the exception of the *noop* action. In its current state, the planner lacks the ability to decide on an exact duration for other actions.

TU-HOP does not output a complete plan that can be shared with other collaborating agents. However, it should be possible to merge together the actions selected in each rollout to form a branching contingent plan that could be shared. Such sharing could then be represented in the planner by increasing the cost of actions that do not correspond to those in the shared plan. This directly models the coordination costs that the group would sustain if the plan were to be changed.

Hindsight optimization is often used with a limited horizon planner. When this is done, it places some responsibility on the heuristic evaluation function used at the leaf nodes of the search to correctly identify promising states. An alternative is to use hierarchical planning, in which a complete plan exists at some level of abstraction, and detailed planning is then done on those parts that are ready for execution. Such an approach has been proposed by Kaelbling and Lozano-Pérez (2011).

Hindsight optimization is an unsound reasoning technique. UCT is a popular sampling-based technique that is sound, in the sense that it is guaranteed to select the optimal action given an infinite number of samples. Eyerich, Keller, and Helmert (2010) compare hindsight optimization with UCT on the Canadian Traveler's Problem. While they find that UCT does indeed converge better in the limit of many samples, hindsight optimization performed well when the methods were given only a moderate number of samples.

Conclusion

Uncertainty is an unavoidable piece of real-world robotic applications. We have shown how hindsight optimization yields a simple and general approach to planning with location, action outcome and temporal uncertainty. While the technique is approximate, it is easy to implement and our results suggest that it can be successful in practice. Its simplicity and capability make it a flexible baseline against which future temporal uncertainty planning research can be compared.

Acknowledgments

This work was supported in part by NSF (grant 0812141) and the DARPA CSSG program (grant D11AP00242).

References

Barreiro, J.; Boyce, M.; Do, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Ong, J.; Remolina, E.; Smith, T.; et al. 2012. Europa: A platform for ai planning, scheduling, constraint programming, and optimization. *Proceedings of the 4th International Competition on Knowledge Engineering for Planning and Scheduling ICKEPS*.

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. Mbp: a model based planner. In *Proc. of the IJCAI01 Workshop on Planning under Uncertainty and Incomplete Information*.

Bonet, B., and Geffner, H. 2001. Gpt: a tool for planning with uncertainty and partial information. In *Proc. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*.

Burns, E.; Benton, J.; Ruml, W.; Yoon, S.; and Do, M. 2012. Anticipatory on-line planning. In *Proceedings of the Twenty-second International Conference on Automated Planning and Scheduling (ICAPS-12)*.

Chong, E.; Givan, R.; and Chang, H. 2000. A framework for simulation-based network control via hindsight optimization. In *IEEE Conference on Decision and Control*.

Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2003. Incremental contingency planning. In *Proceedings of the ICAPS-03 Planning under Uncertainty and Incomplete Information*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence* 49(1):61–95.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-quality policies for the canadian travelers problem. In *Third Annual Symposium on Combinatorial Search SoCS-10*.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems AIPS*, 61–67.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Ingrand, F.; Chatila, R.; Alami, R.; and Robert, F. 1996. Prs: A high level supervision and control language for autonomous mobile robots. In *IEEE Conference on Robotics and Automation ICRA*.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical planning in the now. In *IEEE Conference on Robotics and Automation ICRA*.

Kiesel, S.; Burns, E.; Ruml, W.; Benton, J.; and Kreimendahl, F. 2013. Open world planning for robots via hindsight optimization. In *Proceedings of the ICAPS-13 Workshop on Planning for Robotics (PlanRob-13)*.

Laborie, P., and Ghallab, M. 1995. Ixtet: an integrated approach for plan generation and scheduling. In *INRIA/IEEE*

Symposium on Emerging Technologies and Factory Automation ETFA, volume 1, 485–495 vol.1.

Lemai, S., and Ingrand, F. 2003. Interleaving temporal planning and execution: Ixtet-exec. In *Proceedings of the ICAPS Workshop on Plan Execution*.

Little, I.; Aberdeen, D.; and Thiébaux, S. 2005. Prottle: A probabilistic temporal planner. In *Proceedings of Conference on Artificial Intelligence (AAAI*.

Mausam. 2007. *Stochastic Planning with Concurrent, Durative Actions*. Ph.D. Dissertation, University of Washington.

Mercier, L., and van Hentenryck, P. 2007. Performance analysis of online anticipatory algorithms for large multistage stochastic programs. In *Proceedings of IJCAI*.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, 494–499.

Nilsson, M.; Kvarnstrom, J.; and Doherty, P. 2014. Efficientidc: A faster incremental dynamic controllability algorithm. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS-14)*.

Shani, G., and Brafman, R. 2011. Replanning in domains with partial information and sensing actions. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, 2021–2026.

Wu, G.; Chong, E.; and Givan, R. 2002. Burst-level congestion control using hindsight optimization. *IEEE Transactions on Automatic Control*.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of Conference on Artificial Intelligence (AAAI)*.

Yoon, S.; Ruml, W.; Benton, J.; and Do, M. B. 2010. Improving determinization in hindsight for on-line probabilistic planning. In *Proceedings of the Tenth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-07)*.

A Flexible ANML Actor and Planner in Robotics

Filip Dvořák[†], Arthur Bit-Monnot[*], Félix Ingrand[*], Malik Ghallab[*]
[†]Charles University, Prague
[*]LAAS/CNRS, University of Toulouse

Abstract

Planning in robotics must be considered jointly with Acting. Planning is an open loop activity which produces a plan, based on action models, the current state of the world and the desired goal state. Acting, on the other hand, is a closed loop on the environment activity (to execute command and perceive the state of the world). These two deliberative activities must be integrated and need to handle time, concurrency, synchronization, deadlines and resources. The timeline representation for temporal plan space planning and acting is very expressive; it is also quite flexible for integrating planning and acting. The ANML language is a recent proposal motivated by combining the expressiveness of the timeline representation with the decomposition of HTN methods. This paper reports on FAPE (Flexible Acting and Planning Environment), to our knowledge the first system integrating an ANML planner and actor. Our current focus is not efficient temporal planning per se, but the tight integration of acting and planning, which is addressed by: *(i)* extending HTN methods with refinements, given by PRS procedures, of planned action primitives into low-level commands, *(ii)* interleaving the planning process with acting, the former implements plan repair, extension and replanning, while the latter follows PRS skills refinements, and *(iii)* executing commands with a dispatching mechanism that synchronizes observed time points of action effects and events with planned time.

FAPE has been integrated to a PR2 robot and experimented in a home-like environment. The paper presents how planning is performed and integrated with acting, and describes briefly the robotics experiments and reports on initial performances.

Introduction

Planning is a form of reasoning, through prediction and search, about future changes that can be produced in a system. These changes occur naturally over time. Most contributions to planning abstract away time as state transitions.[1] At an abstract level, this is a legitimate approximation as it simplifies the reasoning. Explicit time is however required in many applications, e.g., for dealing with synchronization with events and others actors, for managing deadlines and time-bounded resources, and for handling concurrency.

Temporal planning comes in two main flavors: extended state space representations and timeline representations. The former is based on states (i.e., snapshots of the entire system) and temporally qualified durations between states. The latter relies on possible evolutions of individual state variables over time (i.e., partial local views of state trajectories), together with temporal constraints between elements of timelines.

Most recent works on temporal planning favor the extended state space representation on the basis of PDDL2.1 (Fox and Long, 2002) with the so-called durative actions. This drive is explained by the wealth of search techniques and domain independent heuristics that have been developed for state space planning, resulting in significant performance improvements. But of a few exceptions, these planners have however a limited handling of concurrency. The timeline alternative representation permits naturally to refer to instants beyond starting and ending points of actions and to handle various kind of concurrency requirements. It is also more flexible in the integration of planning and acting. Timeline planners implement plan-space search algorithms more often than state-space techniques. These algorithms have not scaled up as well as state space planners or HTN planners.

Hierarchical task networks is indeed a planning representation that accounts for numerous deployed applications of significant size. HTN planners benefit from domain specific knowledge expressed as task decomposition methods. In many domains, methods are very naturally formulated. Temporal planning with HTN has not developed as well as with timelines or state space representations.

The Action Notation Modeling Language (ANML) (Smith et al., 2008) is motivated mainly by blending the expressive timeline representation with the decomposition of HTN methods. This paper reports on FAPE, and its planner implementing the ANML language. Our motivation is not efficient planning per se, but the tight integration of acting and temporal planning with task decomposition embedded on a robotic platform. This is addressed by:

- extending planning decomposition methods (Planning)

This work has been conducted within the EU SAPHARI project (http://www.saphari.eu/) funded by the E.C. Division FP7-IST under Contract ICT-287513.

[1]This is also the case for many AI approaches about reasoning on change, e.g., in the LTL, CTL and similar logics, T stands for time, but time is abstracted out.

with refinements of planned action primitives into low-level commands (Acting), these refinements are currently brought by PRS decomposition procedures,

- interleaving the planning process with acting, the former implements plan repair, extension and replanning, while the latter follows PRS refinements,
- executing commands with a dispatching mechanism that synchronizes observed time points of action effects and events with planned time.

The FAPE system currently includes modular components to perform Planning and Acting (as introduced in (Ingrand and Ghallab, 2013)).

FAPE includes a first ANML planner that supports a unique combination of features of least-commitment plan-space planning, explicit time maintained by a sparse simple temporal network and hierarchical task decomposition. There are several motivations for our design choices:

- plan-space planning with least-commitment naturally supports plan repair, which is essential when acting is a concern,
- simple temporal network supports efficient consistency checking and having a sparse network (without saving constraint propagations) allows us to update temporal relations along with the feedback from execution, and
- hierarchical task decomposition allows for highly scalable domain adaptable planning.

FAPE has been integrated to a PR2 robot and experimented with in a real home-like environment. This is a work in progress. A formalization of the planning–acting integration and a full characterization of the performance of the system are beyond the scope of this paper. Its contribution is to present FAPE at the planning, acting, and execution levels, to describe the robotics experiments and report on initial performances. The outline of the rest of the paper follows these steps, preceded by a brief section on the state of the art and an introduction to ANML.

Related work

Numerous planners implements the PDDL2.1 extended state space representation with durative actions, e.g., RPG, LPG, LAMA, TGP, VHPOP and Crickey. Among these planners, COLIN (Coles et al., 2012) is a notable exception that can manage concurrency and even linear continuous change.

The timeline approach goes back to the IxTeT planner (Ghallab and Laruelle, 1994) that reasons on chronicles. A chronicle defines time-points, temporal constraints between its instants, changes in the values of state variables, persistence of these values over time, and atemporal constraints over state variables parameters and values. Other planners such as RAX-PS (Jónsson et al., 2000), ASPEN (Rabideau et al., 1999), Europa/IDEA/T-ReX (Frank and Jónsson, 2003; Muscettola et al., 2002; Rajan et al., 2009) and APSI (Fratini et al., 2011), rely on a similar temporal representation with timelines and tokens representing change and persistence of the values of state variables over time. Some of these timelines are directly connected to actions and percepts (to integrate perception). These systems express temporal constraints in planning operators using the interval algebra. The organization of the planner along agents (IDEA) or reactors (T-ReX) offers a hierarchical representation of the domain. Still the action models representation with compatibilities (temporal constraints over state variables), which tends to spread out the hierarchical decomposition over more than one compatibilities/reactors, makes them tedious to write and difficult to debug.

The HTN approach is implemented into several planners, e.g., Sipe (Wilkins, 1988), SHOP2 (Nau et al., 2003), SIADEX (Castillo et al., 2006). The latter integrates time to HTN planning without handling concurrency.

ANML (Smith et al., 2008) extends the languages used in Europa and ASPEN with recent constructs from PDDL together with HTN task decomposition methods. We are aware of ongoing developments on the basis of this language[2], but to our knowledge, FAPE is the first system including an ANML planner supporting task decomposition and temporal planning.

Several systems integrates planning and acting, in particular with procedure-based approaches to refine actions into lower level commands with systems such as RAP (Firby, 1987) or PRS (Ingrand et al., 1996). Among these systems, Cypress (Wilkins and Myers, 1995) (Sipe & PRS), TCA (Simmons, 1992) (Task Description Language & Aspen) and XFRM (Beetz and McDermott, 1994) are examples relevant for our approach. IxTeT-Exec (Lemai-Chenevier and Ingrand, 2004) and "Configuration Planner" (Di Rocco et al., 2013) are closer to FAPE since they are based on a timeline planner, but without decomposition method.

Representation and ANML

The FAPE planner uses ANML as representation language. ANML is a rich language allowing the user to introduce planning models in a multitude of ways. While the syntactic sugar is important from the perspective of knowledge engineering, let us focus this presentation on the fundamental representations used.

The FAPE planner relies on parametrized state variables, with typed object variables as parameters, and on timelines over these state variables. The advantages of the state variable representation, as in SAS+ (Bäckström and Nebel, 1995) are well known. The state space represented by state-variables is significantly smaller (we cut out unreachable states) and planning algorithms strongly benefit from such reduction as shown in (Helmert, 2009).

ANML allows us to specify the state variables directly in the planning problem definition. Typing is a natural way to reduce the combinatorics of the parameters in operators. FAPE supports typing and single inheritance between types, as illustrated in this simple example (where < denotes inheritance):

```
type Location;
type Gripper < Location {
        boolean empty; };
```

[2] In particular at NASA Ames Research Center

```
type Locatable{
        Location myLocation; };
type Robot < Locatable {
        variable float battery;
        variable Gripper left;
        variable Gripper right; };
type Item < Locatable;
```

The objects of the domain are type instances:

```
instance Location L1, L2, L3;
instance Robot R1;
instance Item I1;
instance Gripper G1, G2;
```

Temporally annotated statements are for example:

```
[start] R1.myLocation := L1;
[50, 70] I1.myLocation == G1 :-> L3;
[end] I1.myLocation == L3;
```

A temporal annotation is either a time point or interval defined by two time points. These can be relative to a context (e.g. an operator, or a planning problem), such as `start`, `end` and `all`, or absolute time points.

According to the definitions given in (Ghallab et al., 2004), we define a temporal statement to be an assertion over the evolution of a parameterized state variable. We consider three type of statements:

- an *event* specifies a change of the value of the state variable. For instance, the ANML statement [t1 , t2] r . myLocation == l1 :−> l2 represents a change of the state variable myLocation(r) from $l1$ to $l2$ between time $t1$ and $t2$, where r, $l1$ and $l2$ are object variables and $t1, t2$ are time points. The value of the state variable is $l1$ at time $t1$ and $l2$ at $t2$; it is unspecified in $]t1, t2[$. An event referring to a single time point is considered as being *instantaneous*, e.g., [t] Switch == On :−> Off indicates a value of the switch as On at time t and as Off right after t.
- a *persistence condition* specifies a constraint on the value of a state variable over an interval. For instance, the ANML statement [t1 , t2] s . myLocation == l3 states that myLocation(s) keeps the value $l3$ over the interval $[t1, t2]$, where s and $l3$ are object variables and $t1$, $t2$ are time points. For the moment, FAPE only handles equality and non-equality constraints.[3]
- an *assignment* is a special case of *event* specifying a new value to a state variable regardless of its previous one. For instance, the ANML statement [t] r . myLocation := l3 states that r will be at location l3 at time t without any condition on its previous location.

Actions are defined as partially instantiated operators that may have several possible decompositions into a partially ordered set of primitive actions. Effects and preconditions are represented as temporally annotated statements occurring between the start and end time of the action. Thus a planing operator is a tuple $(name, maxDuration, P, E, D)$, where $name$ is the unique name of the operator, $maxDuration$ is the function that can be evaluated into a number at the moment of operator application and represents its maximal duration (after which the operator is considered to be failed), P is a set of typed parameters, E is a set of temporal statements and D is a set of decompositions. Parameters of an operator are typed object instances as defined in ANML, they are further used to impose binding constraints between events and decomposition operators. A decomposition is a set of partially ordered and partially instantiated operator references (the action must always occur in the time interval of its parent operator, its parameters are bounded to the values defined in the parent, if any).

```
action Pick(Robot r, Item i, Location l){
  :decomposition{
     PickWithGripper(r, r.left, i, l); };
  :decomposition{
     PickWithGripper(r, r.right, i, l);};};

action PickWithGripper
    (Robot r, Gripper g, Item i, Location l){
  maxDuration := 10;
  [start, end]{ g.empty == true :-> false;
                r.myLocation == l;
                i.myLocation == l :-> g;
              }; };
```

The power of hierarchical decomposition (as in HTN) lies in being able to encode expert level knowledge into the domain by making explicit the various possible decompositions of a task, instead of relying on a search mechanism to find these possible decompositions from basic action models. Of course, this also depends of the skill of the programmer, yet, our experiences with various formalisms indicate that HTN are better suited for planning in robotics. While the refinement of the action can be as simple as the action Pick we have introduced, one can imagine going further, e.g., Transport → TransportByRobot → Move, Pick, Move, Drop, or even PickWithGripper decomposed with motion planning techniques.

FAPE internal structures

FAPE planning and acting components rely on several key data structures that provide efficient handling of state variable evolutions, constraints and plans. In the following subsections we present the timelines, temporal network, constraint network and task network.

Timelines and Chronicles

To capture the information on the evolution of state variables over time, we use timelines with the same semantics as used in (Ghallab et al., 2004, Sec. 14.3). A timeline is a set of temporal statements related to a unique state variable. A timeline Φ is a tuple (x, F, C) where x is a parameterized state variable, F is a set of temporal statements and C is a set of temporal constraints and binding constraints over the time points and object variables in F.

Two essential properties of timelines need to be handled: *consistency* and *causal support*. A timeline (x, F, C) is consistent when the constraints in C are consistent and when no pair of assertions in F are possibly conflicting. Intuitively,

[3] Inequality constraints, e.g., $<, \leq$ etc., will be added together with the management of resources.

two assertions are conflicting when they specify two possibly distinct values of x at the same time. This may happen when the two assertions are allowed to overlap in time with possibly incompatible values (with straightforward cases related to conflicts between persistence, events and mixed conflicts). Additional temporal or binding constraints, called *separation constraints*, may be needed in C to remove possible conflicts and make the timeline consistent.

A timeline (x, F, C) supports an assertion α when there is an assertion $\beta \in F$ that can be used as a *causal support* for α and when α can be added to the timeline consistently. More precisely, when α asserts a persistent value v for x or a change of value from v to v' starting at time t, we require β to establish a value w at a time t' such that $t' < t$ and $w = v$ and that this value can persist consistently until t. Here also additional constraints, i.e., $t' < t$ and $w = v$ and separation constraints, can be needed to make the timeline support α.

We define a *chronicle* as a tuple (T, C) where T is a set of timelines and C is a set of temporal and binding constraints. We say that a chronicle is consistent if each timeline in T is consistent, and the union of constraints in the timelines of T with those of C is consistent.

Temporal Constraint Network

Dealing with explicit time implies taking into account temporal constraints between identified time points of the planning process (such as the beginning of an action or the occurrence of a contingent event). Repairing plans further requires the ability of removing constraints to reflect real events that might be contradictory with our previous knowledge.

Our temporal network manager is based on the *Simple Temporal Problem* introduced by (Dechter et al., 1991). It is encoded as a directed weighted graph in which an edge from t_i to t_j with weight w_{ij} represents the constraint $t_j - t_i \leq w_{ij}$.

Consistency is checked on constraint addition by detecting negative cycles in the graph which is a sufficient and necessary condition of STN consistency. This step is performed by running, upon constraint addition or removal, an incremental Bellman-Ford algorithm as presented in (Cesta and Oddi, 1996). This allows us to efficiently check STN consistency while keeping a sparse network containing only constraints that were explicitly stated, thus allowing us to easily remove constraints from the network.

In general, temporal plans include uncontrollable durations (e.g. the time for the robot to go from the kitchen to the living room may vary between 1 and 2 minutes). These durations should not be squeezed by the planner temporal propagation and we must use an approach which guarantee the dynamic controllability (DC) of the plan. We plan to implement the algorithms proposed in (Morris and Muscettola, 2005) to guarantee that the plan remains DC while squeezing controllable duration as needed.

Binding Constraint Network

While planning, new object variables are created when a new lifted action is inserted into a plan: every parameter of the action gives birth to a new typed object variable. These variables appear either as parameters of state variables or as values of state variables. Separation and causal support constraints on these object variables are managed as a binding constraint network. This constraint network is consistent iff there exists an instantiation of variables such that all equality and non-equality constraints are satisfied. We use AC-3 to maintain the arc-consistency, which is a well-known tradeoff between earliness of the failures and computational performance.

Task Network

A task network is a forest of partially instantiated operators, where the branches represent the conjunction of actions into which an action decomposes. We say that the network is decomposed if all leaves are primitives. A single tree corresponds to the decomposition of a single root action. New trees can be added in the task network when new actions are added in the current plan. This mechanism combines HTN techniques with Plan-Space techniques.

The FAPE planner does not support recursive decomposition methods. Recursive methods raise termination and completeness issues, in addition to complexity issues.

Planning

The planning component of FAPE relies on two mechanisms: task decomposition, as in HTN, and resolver insertion, as in Plan-Space Planning (PSP). A planning problem is defined as a triple (V, O, s_{init}), where V is a set of state variables, O is a set of operators and s_{init} is the initial search node. Since we are in plan-space, we do not define a goal state but an initial search node, which is specified with *(i)* a set of initial statements, giving the initial values of state variable and the expected events and persistences, and *(ii)* the plan objectives. The statements in *(i)* are considered to be causally supported. Those of *(ii)* need to be supported by the plan to be built. They are given as a set of goal statements, temporally qualified with the `end` time point, and/or the task to perform (as in HTN), called here *the seed action*, e.g.,

```
action Seed(){
  :decomposition{
    Transport(anyRobot_, I1, anywhere_, L2);
  };
};
[end] I1.myLocation == L3;
```

In this example, the objective is to achieve the `Transport` task and, at the end to have item `I1` at location `L3`. Note that this specification of the objectives through assertions and a seed action can be redundant, or even inconsistent. It is up to the domain designer to make sure that the domain and problem specification are consistent. While it may be useful to specify goals for one state variable through goal statements and use the seed actions for another state variable, we discourage the domain designer to use both for a single state variable, where the semantics is not clear — there is no syntactical construct to temporarily relate seed actions with goal statements.

The planner search node is a tuple (Φ, T), where Φ is a chronicle and T is the task network. We say that a search node is consistent if both Φ and T are consistent. Planning proceeds by identifying flaws in a search node and iteratively applying resolvers until a search node is reached that is consistent and with no flaws.

Flaws and Resolvers

Planning proceeds as in PSP, by addressing the flaws of a current search node. A search node $n = (\Phi, T)$ may contain the following flaws:

Open goal. An open goal is any statement in Φ that does not have a causal support.

Undecomposed actions. An undecomposed action is a non primitive action appearing as a leaf in the task network; it needs to be decomposed.

Threats are dealt with incrementally through separation constraints, that maintain each timeline consistent, and through causal support constraints.

The resolvers for an undecomposed action flaw are the existing methods specified for its decomposition. Applying a method as a resolver consists in expanding the action node with its specified decomposition with the temporal and binding constraints inherited by the decomposed action.

An open goal α may have two types of resolvers:

- any assertion $\beta \in \Phi$ that can be used to support α; applying such a resolver consists of adding the causal support constraints and the separation constraints required to have α supported.
- an action that provides an assertion β that can be used to support α. Applying such a resolver requires adding the action together with the support constraints and separation constraints.

The newly added action may in turn bring new unsupported statements.

Notice that there are two ways of inserting an action into a partial plan: through a decomposition, or directly by adding an action as a provider of a support for an open goal. The same action may be added as a provider at some point and appear through a decomposition at a later point. A possible redundancy may result from this. The FAPE planner does not currently implement a merging operation over the task network. This will be the object of future work.

Search

Given that a search node π is a solution if it is consistent and with no flaws, search proceeds by identifying flaws of π (i.e. its open goals and undecomposed actions) and applying a resolver for one selected flaw while maintaining the resulting search node consistent. For the purposes of demonstration, we stick, for the moment, to the PSP recursive nondeterministic schema (Ghallab et al., 2004).

The PSP algorithm (See Algorithm 1) at each step of the recursion deterministically chooses a flaw to resolve (selection is done with the simple min-domain heuristic) and then chooses nondeterministically the resolver as follows:

- if the application of a resolver returns a failure then another recursion with a different resolver is performed
- if all resolvers were tried unsuccessfully then a failure is returned to the previous choice point

We can as well modify the non-determinism to reach the optimal solution with regard to some objective function. In practice, our current implementation uses a best-first search strategy, with the number of open goals as a distance evaluation to a consistent search node.

Algorithm 1 Main PSP Algorithm

```
function PSP(π)
    flaws ← OpenGoals(π) ∪ UndecomposedLeaves(π)
    if flaws = ∅ then return (π)
    end if
    select any flaw φ ∈ flaws
    resolvers ← Resolve(φ,π)
    if resolvers = ∅ then return failure
    end if
    nondeterministically choose a resolver ρ ∈ resolvers
    π ← Apply(ρ,π)
    return PSP(π)
end function
```

Acting

In a system like FAPE, Acting and Planning are integrated. Acting, is more complex than just Execution of platform commands. Often, the actions in the plan are still at a too high level to be directly executed on the platform. From our point of view, we consider in FAPE the basic functions relevant to Acting, and introduced in (Ingrand and Ghallab, 2013), to include: refinement, instantiation, time management and coordination, non determinism and uncertainty, plan repair. In the current FAPE implementation, they are all but one (non determinism and uncertainty) handled.

Acting refines online an action into a collection of closed-loop functions, referred to here as skills; a skill processes a sequence (or a continuous stream) of stimulus input from sensors and output to actuators, in order to trigger motor forces and control the correct achievement of chosen actions. We currently use PRS procedures to refine fully instantiated plan actions into motor commands, as well as to perceive the environment and inform the Planner of important changes. PRS skills also provide some local action recoveries for situations where the procedure can handle an alternative way to perform the action (e.g. to consider an alternative grasping pose, or an alternative path to reach a particular location). For our PR2 implementation, the basic motor commands and perception are provided by ROS actions, nodes and also GenoM3 (Mallet et al., 2010) modules. We plan to integrate other skill execution frameworks which can handle different type of acting representation (MDP, DBN, FSM, etc).

For dispatching, fully instantiated and scheduled actions are passed to the Acting component according to their starting time. The planner maintains a partially instantiated plan (only the necessary binding and temporal constraints are applied), which represents a set of valid plans (time and object variables are instantiated when needed). Actions selected for

execution are found by taking the ones whose preconditions are met and whose start times fit in an execution window (e.g. we want to get actions that can be started in the next x seconds). The temporal variables and constraints of those actions are instantiated and the actions are then returned. Further calls instantiate more and more actions while the future instantiation of the actions not yet scheduled is kept as open as possible. Once an action is finished, acting reports the actual end date of its execution. This exact date is then integrated in the current plan, and the temporal propagation, as described in the "Temporal Constraint Network" section, is performed. The action fails if it take less or more time than planned. Such temporal failure is reported to the planner which can then attempt to repair the plan accordingly. Note that in the general case, the acting component can also inform the planner that an action is taking too long, yet, wait for the planner to plan and send an abort action as a result of this problem (the acting component does not take the freedom to abort an action which is running late). An action can also fail because the skill failed (e.g. despite multiple attempts, the robot cannot grasp an object, or reach a location). The acting component then retrieves a description of the changes of the world that occurred and send it to the planner which integrate these "unexpected" state variable transitions in its plan.

Considering we have a plan and one of the actions in the plan fails during the execution, the plan-repair consists of the following steps:

1. Removing the action from the task network.
2. Removing all the statements introduced by the failed action from the timelines which shall generate new flaws.
3. Running the PSP algorithm until the flaws are resolved.

Our repair approach is limited to the removal of just the one failing action, we do not consider cascades of other potential failures. There certainly are cases when the repair does not find a plan and we need to replan, making the repairing a wasted effort. However, most of the time repairing the plan is much faster than replanning and the overall benefit for the responsiveness of a real-time system is significant, as we shall show in the following section.

Experimental Setup and Results

FAPE is designed to be used as an embedded system. The current implementation has been experimented on a PR2 (Figure 1) to plan service robot type of tasks. For example, the PR2 moves around in an apartment and detects objects which are misplaced (e.g. a video tape in the bedroom, or a book on the dining table) picks them up and stores them away in their proper location (respectively by the TV set, and in the bookshelf).

In the current setup, we rely on some of PR2 basic capabilities[4]: navigating in a household like environment; recognizing objects; picking them up and putting them down. Actions are dispatched just in time to PRS which executes them when their start time has arrived. PRS monitors the proper

[4]http://wiki.ros.org/pr2_navigation
http://wiki.ros.org/pr2_tabletop_manipulation_apps

Figure 1: The PR2 Robot.

execution and reports success or failure. In case of failure, the proper relevant state variable values are sent back to the planner as an ANML block which needs to be introduced into the current plan, leading to repair or replan. The implicit behavior of the actor is always to repair the plan, while the replanning is only called once the repair fails completely.

The real-world scenarios we have used were not as demanding as we needed to stress-test the planner, therefore we have generated a spectrum of much larger problems and ran experiments on them. One of the positive discoveries is that the planning time is linearly dependent on the number of objects in the system — problems with hundreds of objects are completely planned in less than a second, thousands of objects do not increase the planning time over ten seconds. This is caused mainly by our non-ground representation, where the combinatorial explosion on operators does not occur. Figure 2 reports on experiments conducted with increasing length of the plan. It should be noted that the planning problem at hand, while realistic in robotics, is simple and cannot be compared to traditional planning benchmarks. As expected, the planning time increases rapidly with the length of the plan up to a 20 actions threshold where the planner no longer finds plan.

To thoroughly test the complete system, we also wrote a simple simulator in PRS which randomly simulates action failures (navigation does not reach the final destination, pickup misses or drops the object, etc), and out of bound time execution (the action takes less or more than the planned duration interval) on a domain with multiple robots. We show that, independently of the number of search nodes generated while producing the initial plan, the FAPE planning component is able to find a trivial repair in a matter of

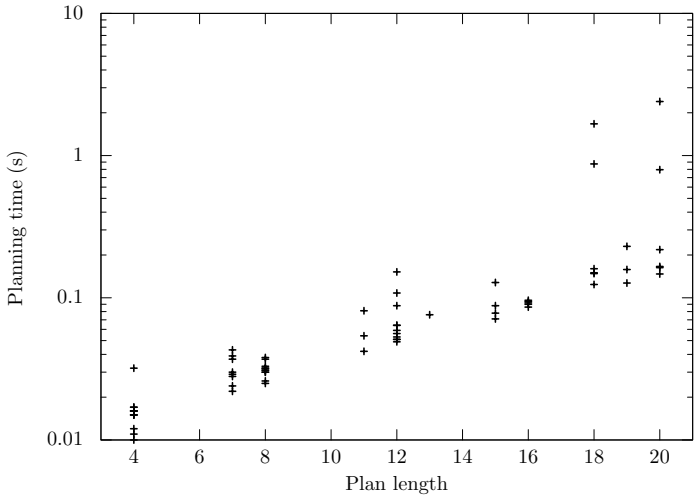


Figure 2: Planning time on simple navigation, pick and place problems with a hundred object constants. Timeout occurred for three instances with a plan length of 20. Over twenty actions in the plan, the planner has a high ratio of timeouts.

	Number of instances
$n_{repair} \leq 15$	102 (82.2%)
$15 < n_{repair} < n_{planning}$	7 (5.7%)
$n_{repair} \geq n_{planning}$	6 (4.8%)
repair failed	9 (7.3%)

Table 1: Number of search nodes generated while repairing the plan (n_{repair}) with respect to the number of nodes generated while producing the initial plan ($n_{planning}$). Over the experiment, $n_{planning}$ has an average value of 319.

milliseconds in the vast majority of cases. Table 1 reports on the number of plan space search nodes explored while repairing a plan compared to the ones explored while generating the initial plan. This result is particularly important as it shows that repairing the plan instead of replanning often saves significant computational effort, which is even more crucial in embedded planning, where the responsiveness of the planner often directly projects into the system performance. Furthermore, repairing the plan allows entities that are not affected by the failure (such as other robots) to keep acting while the plan is being repaired.

Future work

As far as we know, FAPE is the first system including a planner supporting most ANML features – combination of HTN planning and explicit time representation; and plan-space planning. It integrates acting together with planning and both decisional functionalities rely on the same internal representation. Each functionality is critical with regard to the efficiency of the whole system and as such it deserves our attention in future development. The planner shall benefit significantly from the addition of proper resource management similar to the one implemented in IxTeT (Laborie and Ghallab, 1995), a stronger heuristic, as well as the addition of specific and domain dependent planners (e.g. motion or manipulation planner). Meanwhile the Acting system will provide other acting framework than the PRS refinement procedures used for now (e.g. MDP policies (Morisset and Ghallab, 2008), DBN (Infantes et al., 2010), etc). We also plan to implement and compare new models of interleaving planning and acting, where we would concentrate on the decision making between alternative action refinement, repairing and replanning — how to recognize and predict when one is preferred to the other. Similarly, we plan to investigate the inclusion of delayed methods decomposition. The planner, instead of expanding all tasks down to the action leaves may delay and delegate some designated decomposition to the acting component.

We have designed but not yet experimentally tested new control mechanics for decomposition that bring the domain designer more power to fine tune the search and also provide more support for embedded planning. All of the extensions are part of method definition, we call those extensions hard, soft and weak.

The hard extension is an additional condition (a temporal statement) that tells the planner if the method needs to be decomposed (if the condition does not hold then we do not decompose the method and it does not invalidate consistency). The extension allows a multitude of control use-cases to be introduced, e.g. we may start decomposing certain methods only once we get close to their execution — this is the case for the navigation action that can be abstracted as a motion from a to b, until we approach the time of the action and need refine it into a sequence of path following actions that would be otherwise unnecessary to keep in the plan in advance.

The soft extension allows us to define priorities of decompositions — we simply assign a priority to every method then we try expand those with the highest priority first, we can see this extension as an explicit heuristic entered by the domain designed or the real-time environment.

The weak extension represents a look ahead for a decomposition of a method, its main purpose is to propagate new time bounds and constraints. Having a method with several possible decompositions (we call the regular decompositions hard), we add at most one weak decomposition. The weak decomposition method is then always performed when we add the method to the plan and it is non-colliding with any hard decomposition that is chosen later during the search.

We do not directly support conditional decomposition (conditions for each hard decomposition in a method), which can be simulated by using more methods — for each conditional decomposition we instead add a new method having just one decomposition but having the conditional statements as its event, then the original method decomposes into one of the method representing the original conditional decompositions.

While we currently support multi-agent planning (there can be any number of robots in the system that perform their actions in parallel), we are particularly interested in extending the system towards multi-agent planning where actions of some of the entities are not controllable, which shall allow

us to reason and plan human-robot interactions.

Conclusion

We have introduced FAPE, a new framework that integrates Planning and Acting to be embedded in autonomous real-time system such as robots. Using ANML as an input planning langage, we have the expressivity to plan for complex temporal plans with requirements on concurrent actions in dynamic and changing environments and we also allow the user to improve and fine-tune the efficiency of the system by introducing task decompositions which can efficiently prune the search in plan space. We have experimented both Planning and Acting in simulation with large problems and on a PR2 robot which performs service robot type of activities. The development of FAPE continues as a multi-institutional effort to provide a planning/acting system, which we would like to see positioned as a system capturing the state-of-the-art of planning, integrating domain specific planners while maintaining the expressivity of ANML and ease of integration with different type of acting components.

References

Bäckström, C. and Nebel, B. (1995). Complexity Results for SAS+ Planning. *Computational Intelligence*, 11:625–656.

Beetz, M. and McDermott, D. (1994). Improving Robot Plans During Their Execution. In *International Conference on AI Planning Systems*.

Castillo, L., Fdez-Olivares, J., García-Pérez, O., and Palao, F. (2006). Efficiently handling temporal knowledge in an HTN planner. *Sixteenth international conference on automated planning and scheduling, ICAPS*.

Cesta, A. and Oddi, A. (1996). Gaining efficiency and flexibility in the simple temporal problem. *Temporal Representation and Reasoning, International Syposium on*, 0:45.

Coles, A. J., Coles, A., Fox, M., and Long, D. (2012). COLIN: Planning with Continuous Linear Numeric Change. *J. Artif. Intell. Res. (JAIR)*, 44:1–96.

Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95.

Di Rocco, M., Pecora, F., and Saffiotti, A. (2013). When robots are late: Configuration planning for multiple robots with dynamic goals. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, pages 9515–5922. IEEE.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of the sixth National conference on Artificial intelligence*, pages 202–206. Seattle, WA.

Fox, M. and Long, D. (2002). PDDL 2.1 : An Extension to PDDL for Expressing Temporal Planning Domains. *Technical Report, University of Durham, UK*.

Frank, J. and Jónsson, A. K. (2003). Constraint-Based Attribute and Interval Planning. *Constraints*, 8(4).

Fratini, S., Cesta, A., De Benedictis, R., Orlandini, A., and Rasconi, R. (2011). APSI-based deliberation in Goal Oriented Autonomous Controllers. In *11th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*.

Ghallab, M. and Laruelle, H. (1994). Representation and Control in IxTeT, a Temporal Planner. In *International Conference on AI Planning Systems*, pages 61–67.

Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated Planning: Theory and Practice.* Morgann Kaufmann.

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6):503–535.

Infantes, G., Ghallab, M., and Ingrand, F. (2010). Learning the behavior model of a robot. *Autonomous Robots Journal*, pages 1–21.

Ingrand, F., Chatilla, R., Alami, R., and Robert, F. (1996). PRS: a high level supervision and control language for autonomous mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 43–49.

Ingrand, F. and Ghallab, M. (2013). Robotics and Artificial Intelligence: a Perspective on Deliberation Functions. *AI Communications*, 27:63–80.

Jónsson, A. K., Morris, P. H., Muscettola, N., Rajan, K., and Smith, B. (2000). Planning in Interplanetary Space: Theory and Practice. In *International Conference on AI Planning Systems*.

Laborie, P. and Ghallab, M. (1995). Planning with Sharable Resource Constraints. In *International Joint Conference on Artificial Intelligence*.

Lemai-Chenevier, S. and Ingrand, F. (2004). Interleaving Temporal Planning and Execution in Robotics Domains. In *Proceedings of the National Conference on Artificial Intelligence*.

Mallet, A., Pasteur, C., Herrb, M., Lemaignan, S., and Ingrand, F. (2010). GenoM3: Building middleware-independent robotic components. In *IEEE International Conference on Robotics and Automation*, pages 4627–4632.

Morisset, B. and Ghallab, M. (2008). Learning how to combine sensory-motor functions into a robust behavior. *Artificial Intelligence*, 172(4-5):392–412.

Morris, P. H. and Muscettola, N. (2005). Temporal dynamic controllability revisited. In *National Conference on Artificial Intelligence*, pages 1193–1198.

Muscettola, N., Dorais, G., Fry, C., Levinson, R., and Plaunt, C. (2002). IDEA: Planning at the Core of Autonomous Reactive Agents. In *Proceedings of the AIPS Workshop on On-line Planning and Scheduling*.

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN Planning System. *J. Artif. Intell. Res. (JAIR)*, 20:379–404.

Rabideau, G., Knight, R., Chien, S., Fukunaga, A., and Govindjee, A. (1999). Iterative Repair Planning for Spacecraft Operations in the ASPEN System. In *International Symposium on Artificial Intelligence, Robotics and Automation for Space*.

Rajan, K., Py, F., McGann, C., Ryan, J. P., O'Reilly, T., Maughan, T., and Roman, B. (2009). Onboard Adaptive Control of AUVs using Automated Planning and Execution. In *International Symposium on Unmanned Untethered Submersible Technology (UUST)*, Durham, NH.

Simmons, R. (1992). Concurrent planning and execution for autonomous robots. *Control Systems, IEEE*, 12(1):46–50.

Smith, D. E., Frank, J., and Cushing, W. (2008). The ANML Language. *The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Wilkins, D. E. (1988). *Practical Planning*. Extending the Classical AI Planning Paradigm. Morgan Kaufman.

Wilkins, D. E. and Myers, K. L. (1995). A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6):731–761.

HATP: An HTN Planner for Robotics

Raphaël Lallement[1,2] and **Lavindra de Silva**[1] and **Rachid Alami**[1]

[1]CNRS, LAAS,
7 avenue du Colonel Roche,
F-31400 Toulouse, France

[2]Univ de Toulouse, INSA, LAAS,
F-31400 Toulouse, France

Abstract

Hierarchical Task Network (HTN) planning is a popular approach that cuts down on the classical planning search space by relying on a given hierarchical library of domain control knowledge. This provides an intuitive methodology for specifying high-level instructions on how robots and agents should perform tasks, while also giving the planner enough flexibility to choose the lower-level steps and their ordering. In this paper we present the HATP (Hierarchical Agent-based Task Planner) planning framework which extends the traditional HTN planning domain representation and semantics by making them more suitable for roboticists, and treating agents as "first class" entities in the language. The former is achieved by allowing "social rules" to be defined which specify what behaviour is acceptable/unacceptable by the agents/robots in the domain, and interleaving planning with geometric reasoning in order to validate online–with respect to a detailed geometric 3D world–the human/robot actions currently being pursued by HATP.[1]

Introduction

Real-world robotics domains and problems offer natural testbeds for HTN (Hierarchical Task Network) planning. The intuitive hierarchical representation used by such planners allows the often available expert knowledge about a domain to be included with relative ease to guide the search process. This guidance might be abstract steps detailing how a task, such as cleaning a table full of different types of objects, should be performed by the robot, with sufficient flexibility over the more detailed steps and states—e.g. the final locations of objects on the shelf. In practice, the inclusion of such search control knowledge makes HTN planning faster than classical planning, which is particularly important when dealing with robots as they need to be responsive to environmental changes involving other robots, and more importantly, humans.

In this paper we describe the HATP (Hierarchical Agent-based Task Planner) HTN planner and show how it is particularly suited for use in robotics. HATP is based on SHOP (Nau et al. 1999), but unlike this planner and other HTN planners such as Nonlin (Tate 1976), SHOP2 (Nau et al. 2003) and UMCP (Erol, Hendler, and Nau 1994), HATP offers a user-friendly domain representation language inspired by popular programming languages, making it easier for roboticists, and indeed computer scientists alike, to become quickly acquainted with the syntax and semantics. We give insights into a formal mapping from this HATP language into an equivalent classical representation, but leave the detailed treatment for a separate paper.

An important feature of HATP is that it treats agents as "first-class entities" in the domain representation language. It can therefore distinguish between the different agents in the domain as well as between agents and the other entities such as tables and chairs. This facilitates a post-processing step in HATP that splits the final solution (sequence of actions) into multiple synchronised solution streams, one per agent, so that the streams may be executed in parallel by the respective agents by synchronising when necessary.

The planning algorithm of HATP has also been extended in various ways. First, it incorporates a simple mechanism to take into account the (user-defined) cost of executing actions, so that instead of returning the first arbitrary solution found, it keeps searching until an optimal (least-cost) one is found.[2] Second, HATP has been extended to be more suitable for Human-Robot Interaction (HRI); in particular, "social rules" can be included by the user to define what the acceptable (and unacceptable) behaviours of the agents are. Two examples are: what sequences of steps should be avoided in final solutions, and a limit on the amount of time a person should spend waiting (and doing nothing). The rules are then used to filter out the primitive solutions found that do not meet the constraints.

Finally, there is much ongoing work on interleaving HATP with geometric planning algorithms, so as to validate online the actions being pursued by HATP, by consulting its geometric counterpart. This results in motion planning being performed by the geometric planner to check if the HATP action being planned is actually feasible in

[1]This work has been conducted within the EU ARCAS project (http://www.arcas-project.eu/) funded by the E.C. Division FP7-IST under Contract ICT 287617. We thank the anonymous reviewers for their feedback. The second author has now moved to The University of Nottingham, Nottingham, UK.

[2]The notion of optimality here is "local": HATP finds an optimal solution only from the set of HATP solutions obtained using the given methods.

the real world, modelled in great detail via the Move3D (Siméon, Laumond, and Lamiraux 2001) simulation environment. This integration takes an important step towards interfacing HATP's AI planning algorithms and techniques with the planning algorithms and techniques more commonly used by roboticists. In this paper we summarise all of these extensions to HATP, and explicate how they make HATP particularly suited for the Robotics community.

HTN Planning

While classical planners such as STRIPS focus on achieving some goal state, Hierarchical Task Network (HTN) planners focus on solving *abstract tasks*. We have found HTN planning to be particularly useful for robotics applications, as it allows—the often available—instructions from the domain expert to be included in the domain as an intuitive hierarchy. This helps guide the search, making it faster in general than classical planning approaches, and thereby also more practical for real robots that need to be responsive to environmental changes.

The Hierarchical Agent-based Task Planner (HATP) is based on the popular "totally-ordered" HTN planning approach, which unlike "partially-ordered" HTN planning allows calls to external functions—a necessity in our work. This is also highlighted as a feature in the SHOP (Nau et al. 1999) planner, on which HATP is based. The rest of this section focusses on totally-ordered HTN planning.

We define an HTN *planning problem* as the 3-tuple $\langle d, s_0, \mathcal{D} \rangle$, where d, the "goal" to achieve, is a sequence of primitive or abstract tasks, s_0 is the initial state, and $\mathcal{D}$ is an HTN *planning domain*. An operator is as in classical planning, and actions are ground instances of operators. We generally use the terms operator and action interchangeably in this paper. An HTN planning domain is the pair $\mathcal{D} = \langle \mathcal{A}, \mathcal{M} \rangle$ where $\mathcal{A}$ is a finite set of operators, and $\mathcal{M}$ is a finite set of HTN *methods*. A method is a 4-tuple consisting of: the name of the method, the abstract task that it needs to solve, a precondition specifying when the method is applicable, and a *body* realising the "decomposition" of the task associated with the method into more specific subtasks. Specifically, the method-body is a sequence of primitive and/or abstract tasks.

The HTN planning process works by selecting applicable methods from $\mathcal{M}$ and applying them to abstract tasks in d in a depth-first manner. In each iteration, this will typically result in d becoming a "more primitive" sequence of tasks. The process continues until d has only primitive tasks left, which map to action names. At any stage during planning if no applicable method can be found for an abstract task, the planner essentially "backtracks" and tries an alternative method for an abstract task refined earlier.

In more detail, the main steps of the HTN planning process are the following: in each iteration all ground instances are found of the methods available to decompose a chosen task from task network d; one such method instance is chosen arbitrarily that is applicable (whose precondition holds) in the current state of the world; and the instance is applied to d by basically replacing the chosen task with the subtasks in the method's body. The planner backtracks to choose an alternative method instance to one that was previously applied to d only if that method instance did not eventually allow a complete (and successful) decomposition of the top-level goal task(s).

Features of HATP

In this section we present our own encoding in HATP of the Dock-Worker Robots domain (Nau, Ghallab, and Traverso 2004). In this domain, there is a robot (R1) that can move and carry containers, and two crane-agents (K1 and K2) that can lift and put down containers. Furthermore, there are two locations (L1 and L2), each containing two piles (P11 and P12 at L1, and P21 and P22 at L2) that can hold containers. The goal is to place the two containers C1 and C2 on piles P21 and P22, respectively.

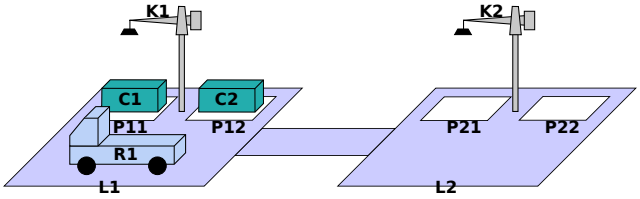


Figure 1: A planning problem in the Dock-Worker Robot domain: robot R1 has to carry the container C1 from P11 to P21, and the container C2 from P12 to P22.

World state representation

In addition to having the standard advantages of total-order HTN planning, HATP also provides an intuitive object-oriented-like syntax for representing and manipulating the world state. This allows roboticists and computer scientists alike to quickly get acquainted with the syntax and start developing HATP domains.

The HATP world specification is defined as a collection of entities, which represent the agent types and object types in the world. This distinction between agents and other objects is important. Agents are treated as first class entities in the language of HATP; moreover, different types of agents may be defined by simply instantiating the default Agent entity. This distinction also facilitates a post-processing step in HATP, which splits the final solution into separate subsolutions to be executed by the respective agents.

An entity has a set of attributes, where an attribute either represents a data value, or a relation between the entity and other entities. For example, a robot-agent may have an attribute carry of type Container indicating that the robot can carry objects of type Container. HATP supports some of the standard data types found in programming languages, such as integers and strings, and also allows defining sets of objects, which are manipulated using the standard set operations. An example of an HATP world specification is shown in listing 1.

```
define entityType Crane, Location, Pile, Container;

define entityAttributes Agent {
  //An agent can be of type Robot or Crane
  static atom string type;

  //For cranes
  static atom Location attached;

  //For robots
  dynamic atom Location at;
  dynamic atom Container carry;
  dynamic atom bool loading;
}

define entityAttributes Location {
  static set Location adjacent;
  dynamic atom bool occupied;
}

define entityAttributes Pile {
  static atom Location attached;
  dynamic set Container contains;
  dynamic atom Container top;
}

define entityAttributes Container {
  dynamic atom Location in;
  dynamic atom Container on;
}
```

Listing 1: HATP entities for the Dock-Worker Robot domain in figure 1. There are five entity types: Agent (default entity), Crane, Location, Pile and Container. The initial value assigned to a static attribute cannot change during planning, whereas a dynamic attribute can be assigned different values over the course of planning. An attribute classified as an atom can only have one value, whereas one classified as a set can have a set of values. The type of an attribute can be any of the primitive types allowed as well as an entity.

The HATP initial world state is then an instantiation of the defined entities, along with value assignments to their attributes. An example of an HATP initial world state is shown in listing 2. Notice that attributes of entities generally map to predicate symbols in standard "classical" initial states, and the entities and values to the parameters of the predicate. For example, K1.attached = L1 could map to predicate attached(K1,L1) in a classical initial state, R1.loading = false to ¬loading(R1), and R1.carry = NULL could be represented in the classical initial state by not including any positive literal in it that has predicate symbol carry, with R1 as its first parameter.

```
R1, K1, K2 = new Agent;
L1, L2 = new Location;
P11, P12, P21, P22 = new Pile;
C1, C2 = new Container;

R1.type = "ROBOT";
R1.at = L1;
R1.carry = NULL;
R1.loading = false;

K1.type = "CRANE";
K1.attached = L1;

K2.type = "CRANE";
K2.attached = L2;

L1.adjacent <<= L2;
L1.occupied = true;
L2.adjacent <<= L1;
L2.occupied = false;

P11.attached = L1;
P11.contains <<= C1;
P11.top = C1;
P12.attached = L1;
P12.contains <<= C2;
P12.top = C2;
P21.attached = L2;
P22.attached = L2;

C1.in = L1;
C1.on = NULL;
C2.in = L1;
C2.on = NULL;
```

Listing 2: An HATP initial state for the Dock-Worker Robot domain. After instantiating the entity types, their attributes are assigned initial values. Note that symbol "<<=" is used to add the element on its RHS to the set on its LHS.

Domain representation

As in standard HTN planning, an HATP domain consists of a set of methods and a set of operators. These are written similarly to traditional HTN domains with the exception where the HATP language offers some user-friendly constructs for defining preconditions of methods and operators, bodies of methods and effects of operators. In particular, variables are defined in HATP methods, and their bindings controlled, via the following constructs; examples of their use can be found in listing 3.

- SELECT binds the given variable in the usual way. In essence, the construct amounts to a "backtrack point" that allows all values of the associated variable—and thereby all ground instances of the method—to be considered.
- SELECTORDERED binds the variable in some given order, governed by a user-supplied ordering relation. Moreover, the variable can be bound in ascending or descending order with respect to the relation.
- SELECTONCE binds the variable only once—the remaining bindings are disregarded. This offers a reduction in the branching factor at the expense of completeness, as some of the ignored bindings may also yield HATP solutions.

While the last construct may result in the loss of HATP solutions, this heuristic is useful in domains where if a solution pursued by taking one binding of the variable—and applying the resulting ground instance of the HATP method—turns out to not work, then no other binding for that variable will work either. For example, imagine a slightly different Dock-Worker Robot domain/problem that has multiple robots, and where taking the shortest path during navigation is not important. This means that if one robot cannot navigate from one location to another, then none of the others will be able

to either. Therefore, there is no need to consider all possible robot-agent bindings as done in listing 3: a single binding will be sufficient.

```
method Transport(Container C, Pile Target) {
  // do nothing if container is in target pile
  empty{C.in == Target;};
  {
    preconditions {
      // container not already in target location
      EXIST(Pile Source2, {C.in == Source2;},
            {Source2.attached != Target.attached;});
    };
    subtasks {
      S = SELECT(Pile, {C.in == S;});
      R = SELECTORDERED(Agent, {R.type == "ROBOT";},
          distance(R.at, S.attached), <);
      K1 = SELECT(Agent,
           {K1.type == "CRANE"; K1.at == S.attached;});
      K2 = SELECT(Agent,
           {K2.type == "CRANE"; K2.at == Target.attached;});
      1: GetReady(R, C, S);
      2: LoadRobot(K1, R, C)>1;
      3: NavFromTo(R, S.attached, Target.attached)>2;
      4: UnloadRobot(K2, R, C)>3;
      5: Put(K2, C, Target)>4;
    };
    ...
}
```

Listing 3: Part of an HATP method to move a container from a source pile to a target pile in a different location. Note that distance is a user-supplied ordering relation; "$<$" means that the variable bindings should be in descending order; and "$N_1 : T > N_2$;" means that task T (labelled N_1) must precede the task labelled N_2.

Observe from listing 3 that, as expected, the subtasks within the method's body are totally ordered. HATP, however, also allows partially ordering subtasks; this is achieved by not specifying ordering constraints between some (or all) of the tasks in the method's body. For example, removing constraint "> 2" from the method in listing 3 would then not require that the task with label 3 occur after the one with label 2. Note that such partial ordering of tasks is merely a convenience: it is an alternative to supplying multiple totally-ordered methods corresponding to every possible linearisation of the partially ordered subtasks. This is exactly what happens during planning: the set of partially ordered subtasks in a method's body is handled by taking all possible linearisations of the set, essentially creating additional HATP method options to consider for the parent task's decomposition. Since partially ordering subtasks results in an exponential increase in the number of method options, it should be used with appropriate care. Introducing "true" partially-ordered planning into HATP is left as future work: the algorithms are not obvious as we want to have the ability to use evaluable predicates in preconditions, for which maintaining the complete state of the world at each step of the planning process is the obvious solution (Nau et al. 1999).

Some other useful constructs supported by HATP are EXIST, IF, and FORALL. As in other HTN planners such as SHOP, construct EXIST is used only in preconditions of methods and operators; IF only in the effects of operators; and FORALL in both preconditions of methods and operators, as well as in the effects of operators. Examples of how these constructs may be used are shown in listings 3 and 4.

```
action Move(
  Agent R, Location From, Location To, Location FinalDest) {
  preconditions {
    R.type == "ROBOT";
    To >> From.adjacent;
    R.at == From;
    To.occupied == false;
  };
  effects {
    R.at = To;
    From.occupied = false;
    To.occupied = true;
    R.path <<= To;
    IF{From !>> R.path;}{R.path <<= From;}
    IF{To.isForbiddenBy == R;}{To.isForbiddenBy = NULL;}
    IF{To == FinalDest;}{
      FORALL(Location LocP,
        {LocP >> R.path;},{R.path =>> LocP;});
    }
  };
  cost{costToMove(From, To)};
}
```

Listing 4: An HATP operator. The expression "$A >> B.attr$" holds if element A is in the set $B.attr$, and the expression's negation is specified using "$A! >> B.attr$". Expression "$B.attr <<= A$" adds element A to set $B.attr$, and "$B.attr =>> A$" removes A from $B.attr$.

Plan production

HATP is able to find the least-cost primitive solution that solves the goal task(s) at hand, as done for example in (Nau et al. 2003). To this end, HATP keeps track of the least-costly plan computed so far, as well as the total cost of the current partial plan being pursued, and then avoids adding any action to it that will definitely lead to a costlier partial plan. Indeed, in the worst case this requires looking through all HATP solutions for the given goal task(s). Moreover, since the HATP search space is governed by the methods supplied, there may be other low-cost solutions (corresponding to methods not supplied) that HATP does not take into account.

The cost of the partial plan is computed via "cost functions". A cost function is a user-supplied C++ function that is linked to an HATP operator as shown at the bottom of listing 4. The function can perform any arbitrary calculation to estimate the cost of executing the action; however, for efficiency reasons the function should terminate quickly. An example of such a function is one that computes the cost of executing an action to send data from one robot to another. This might involve checking how much data needs to be sent and thereby how much time it would take to do the transfer.

By using cost functions associated with the sequence of primitive actions pursued so far, HATP determines the total cost of the sequence, and avoids pursuing it further if by adding the next action the total cost would exceed the cost of the lowest-cost solution found so far.

Once HATP finds a solution—a sequence of primitive actions—it then splits the solution into multiple "streams",

one per agent in the domain, and adds causal links between streams for synchronisation (Alami et al. 2011). To determine which actions in the final solution belong to which agents, the HATP language reserves the first variable of every operator's name: it must always bind to the name of the agent responsible for eventually executing the operator. The second and subsequent variables of an operator's name may also be used as placeholders for agent names if necessary. Such an operator would then be a "joint operator": one that needs to be executed in parallel by all the robots/agents that it refers to.

Once the different streams are separated, they may then be executed. The stream (if any) belonging to the agent that formulated the plan may be executed by the agent directly, whereas actions in other streams need to be delegated to their respective agents, and the environment monitored to determine if the actions were successfully executed. Figure 2 shows a plan produced for the Dock-Worker Robots problem depicted in figure 1 with different streams belonging to the different agents in the domain.

Note that in the case of joint operators, all the agents involved need a "stronger" synchronisation than what causal links entail. For instance in a robot-robot synchronisation they may need to set some rendezvous points so as to exchange information just before starting. This may also involve visual servoing, both in robot-robot and human-robot joint operators.

HATP in an HRI context

As highlighted by (Alili, Alami, and Montreuil 2009) one challenge in robotics is to develop socially interactive and cooperative robots. The meaning of socially interactive robots is defined in (Fong, Nourbakhsh, and Dautenhahn 2003) which states that they must "operate as partners, peers or assistants, which means that they need to exhibit a certain degree of adaptability and flexibility to drive the interaction with a wide range of humans". (Klein et al. 2004) implemented that in what they called "ten challenges for human robot teamwork". We are convinced that task planners can take care of several of these challenges. In this case the robot should be able to (Klein et al. 2004): *(1)* signal in what tasks it can/wants to participate; *(2)* act in a predictable way to ensure human understanding of what it is doing; *(3)* publicise its status and its intentions; *(4)* negotiate on tasks with its human partner in order to determine roles and decide how to perform the tasks; and *(5)* deal with social conventions, as well as its human partner's abilities and preferences.

To address some of those challenges HATP includes mechanisms to filter plans so as to keep only those suitable for HRI. To this end, HATP allows the specification of the following filtering criteria.

Wasted time: Avoids plans where an agent(s) mentioned in a plan spends a lot of its time waiting.

Effort balancing: Avoids plans where efforts are not fairly distributed among the agents mentioned in a plan.

Control of intricacy: Avoids plans with too many interdependencies between the actions of agents mentioned in the plan, as a problem with executing just one of those actions could invalidate the entire plan.

Undesirable sequences: Avoids plans that violate specific user-defined sequences.

Combining some of the above criteria could help yield the following interesting behaviours: *(1)* the human ends up doing a lot of the tasks, but yet the overall effort (Alili, Alami, and Montreuil 2009) taken to do them is significantly lower than what the robot puts to do a lower number of effort-intensive tasks; *(2)* avoiding, when possible, having the human wait for the robot several times, which essentially prevents the streams from having too many causal links between them. The filtering criteria are implemented by looking through all the plans produced and filtering out the ones that do not meet the requirements specified. In the future we intend to study algorithms that do such filtering online, rather than after primitive solutions are found.

Interleaving with geometric reasoning

While an HTN hierarchy allows one to intuitively reason about high-level tasks such as Transport in terms of more specific tasks, and eventually in terms of basic actions, these still "abstract out" the lowest possible level of detail by making certain assumptions about the world. For example, HATP operator Move in listing 4 assumes that as long as location To is adjacent to location From, and To is not occupied, that the robot at From will be able to navigate to location To. Clearly, this may not always work for various reasons, such as there being an obstacle in the path, or certain geometrical characteristics of the robot and the connecting path making the move physically impossible. Combining HATP—and symbolic/task planning in general—with the geometric planning algorithms used in robotics is therefore essential to be able to obtain primitive solutions that are viable in the real world.

The work in (de Silva, Pandey, and Alami 2013; de Silva et al. 2014; 2013) proposes an interface between HATP and a geometric planner. This interface is mainly provided via "evaluable predicates"—predicates in HATP preconditions that are evaluated by calling associated external procedures. Such a predicate evaluating to $true$ amounts to a geometric solution existing for the "geometric task" that the predicate represents, and evaluating to $false$ amounts to the non-existence of such a solution. For example, the precondition of an HATP action that gives an object to a person might have an evaluable predicate that invokes the geometric planner to check the feasibility of the task of giving the object to the person, and to store the resulting geometric trajectory if any. This notion of a geometric task is something that is both important in order to have a meaningful link between the two planning approaches, and also specific to the type of geometric planner used. A geometric task essentially corresponds to one or more motion planning goal-configurations, computed (automatically) by the geometric planner by taking into account various criteria such as the visibility and reachability of objects from the perspectives of different robots and humans in the domain. Aptly called Geometric Task Planner (GTP) (Pandey et al. 2012), this planner liberates

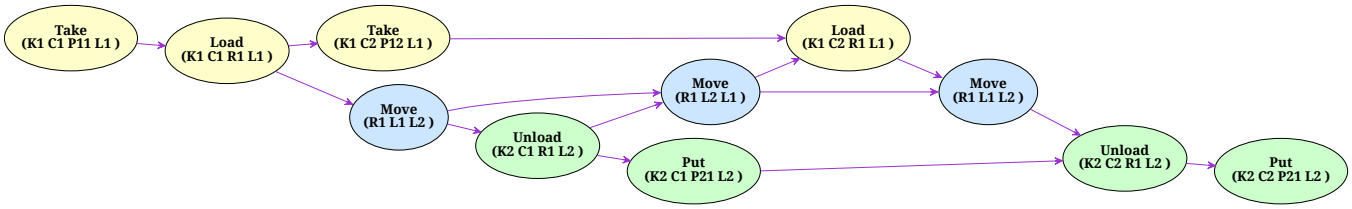


Figure 2: The HATP solution for the DWR problem. There are three streams corresponding to the actions belonging to the three agents, ordered using causal links (shown as arrows). The yellow stream represents the actions of the first crane, the green the actions of the other crane, and the blue the actions of the robots.

HATP from having to reason in terms of low-level details such as grasps and orientations. Using this particular planner for forming the link with HATP also makes the interface proposed different to other interfaces in the literature, such as (Dornhege et al. 2009b; 2009a; Karlsson et al. 2012; Lagriffoul et al. 2012).

The interface between HATP and the GTP is used to interleave their planning algorithms. In one approach, whenever the GTP is invoked by HATP while testing an evaluable predicate, the non-existence of a GTP solution for the associated geometric task (from the current geometric world state) does not lead to the predicate evaluating to $false$; instead, the GTP backtracks to try alternative solutions for the previously invoked geometric tasks in an effort to make a solution possible for the most recently invoked one. Since this may cause changes to intermediate geometric world states, this approach comes with mechanisms to ensure that such changes do not affect the symbolic world state in a way that invalidates the HATP plan being pursued. Such mechanisms are, however, not necessary in the second approach to interleaved planning that the authors present. Here, whenever the GTP cannot find a solution for a geometric task, it does not—as before—backtrack to find alternatives for previous geometric tasks, but instead immediately returns with "failure". If this leads to HATP backtracking, HATP then has the option to try, intuitively, a different "instance" of the action that needs to be "undone" as a consequence of the backtrack (in addition to the standard option of trying different *actions*); this different "instance" is basically the same HATP action that needs to be undone, but this time with a different geometric solution attached to it.

An interesting feature of the GTP is its ability to plan not just the robots' tasks/actions but also the humans', by taking into account their respective kinematic models. This makes way for the multiple robots/agents defined in an HATP domain to have a clear association with those defined in the GTP domain. For example, figure 3 shows a simplified library domain (de Silva et al. 2014) where a PR2 robot serves a human customer, consisting of both human and robot actions. While the PR2-actions will be planned by the GTP from the perspective of the PR2 (using its kinematic model), those of the human, which involve paying and taking a book, will be planned from the human's perspective.

The planning and execution architecture

Both HATP and the GTP are part of the larger LAAS robotics architecture (Fleury, Herrb, and Chatila 1997; Alami et al. 2011). This architecture has many components. It uses the Move3D (Siméon, Laumond, and Lamiraux 2001) motion and manipulation planner for representing the robot's version of the real world in 3D and for doing geometric task planning. Through various sensors the robot can also update its 3D world state in real-time. To this end, a tag-based stereo vision system is used for object identification and localisation, and a Kinect (Microsoft) sensor for localising and tracking the human. The execution controller—the Procedural Reasoning System (PRS)(Ingrand et al. 1996)—is responsible for invoking HATP when a task needs to be planned, and also executing the resulting primitive solution returned by HATP by invoking various actuators via the interface provided by Genom (Fleury, Herrb, and Chatila 1997) to the low-level controllers, which is also the framework used to wrap them into individual well-defined modules.

In the current architecture, PRS receives goals from the environment, which it validates by checking for things such as whether the goal has already been achieved. If the goal is valid, it is sent as a task to HATP. If HATP (possibly together with the GTP) successfully returns a solution, it is then executed by PRS, by directly executing the robot's actions and indicating in the right order to other agents, via a dialogue module, what actions they need to execute. To execute an action directly, PRS sends requests to the relevant Genom modules which may result in the robot or an arm moving, for example. Indeed, the Genom modules may actually execute the trajectories found and stored by the GTP if it was invoked by HATP during the planning process. PRS is also able to confirm whether the robot's actions and those of the other agents were successfully executed, by examining the current state of the (symbolic and geometric) world.

Conclusion and future work

We have described in this paper the HATP HTN planner, which has been used extensively for practical robotics applications in the LAAS architecture (Alami et al. 1998) over many years (Alili, Alami, and Montreuil 2009; Guitton, Warnier, and Alami 2012; Warnier et al. 2012; de Silva, Pandey, and Alami 2013; de Silva et al. 2014). We have focussed on describing how HATP is suited for not just HTN planning but also planning in the context of Human-Robot

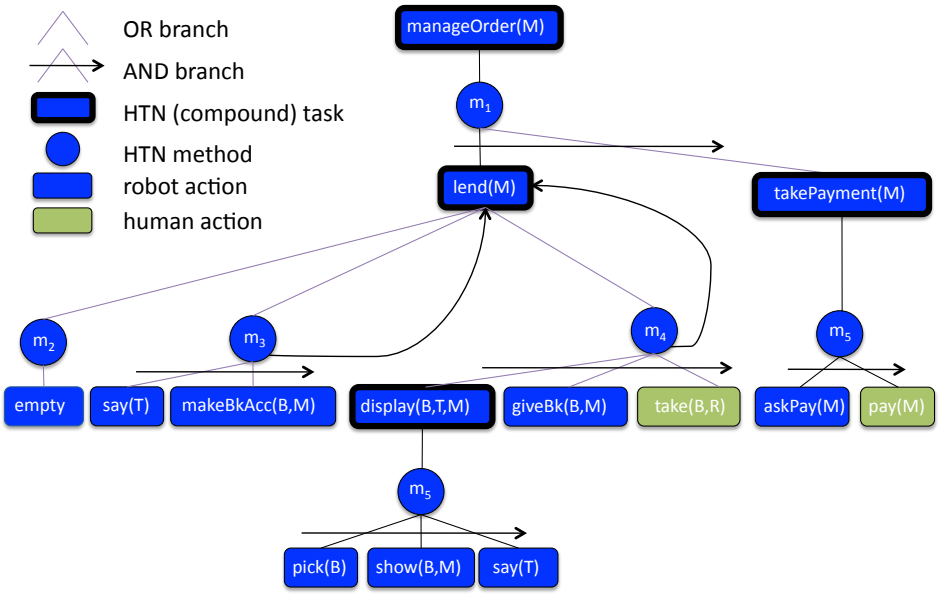


Figure 3: This figure depicts a simplified version of the library domain in (de Silva et al. 2014). Members (M) reserve library books (B) online and then come in person to pick them up from the PR2 (R). The PR2 manages the order placed for the books by recursively lending all the books ordered until there are no more books to lend—in which case it will stop the recursion by choosing method m_2—and then taking payment from the member. Books can either be made accessible on the table (makeBkAcc) via method m_3, or displayed to the member and then given to his/her hand via methods m_5 and m_4. Action say involves speaking out the title (T) of the book.

Interaction, in a multi-agent setting consisting of multiple humans and robots. This was based on two main extensions to HATP: the ability to handle user-supplied "social rules" that specify what is appropriate behaviour for the agents in the domain; and interleaving the HATP planning algorithm with geometric planning algorithms from the robotics community. We have also presented the advantages of the user-friendly syntax and semantics of HATP using our own encoding of the Dock Worker Robot domain described in (Nau, Ghallab, and Traverso 2004).

There has also been some initial efforts toward extending HATP to support separately modelling the beliefs of the different agents in the domain (Alami et al. 2011). This allows reasoning about what the different agents know, including finding conflicting beliefs, and synchronising beliefs by planning to notify agents when there are inconsistencies between their beliefs. Other interesting work on HATP that is currently underway is formalising its domain representation language to show its relation with more traditional representations such as that used by the SHOP planner (Nau et al. 1999). Indeed, this involves developing a mapping from the syntax and notions of HATP to PDDL-like syntax and notions.

In terms of the link between HATP and the GTP, it would be interesting to compare the two different combined backtracking strategies. As the authors in (de Silva et al. 2014) have stated, this would require completing the implementation of the system presented in (de Silva, Pandey, and Alami 2013) so that it may be compared empirically with the system in (de Silva et al. 2014). An analytical evaluation would also be useful to understand in what situations/domains one combined backtracking approach should be favoured over the other. Finally, modifying HATP to interleave planning with execution to make HATP more "responsive" to changes in the environment would make it even more suitable for real-world robotics applications (de Silva et al. 2014).

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. In *Internatonal Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming*, volume 17, 315–337.

Alami, R.; Warnier, M.; Guitton, J.; Lemaignan, S.; and Sisbot, E. A. 2011. When the robot considers the human... In *Proceedings of the 15th International Symposium on Robotics Research*.

Alili, S.; Alami, R.; and Montreuil, V. 2009. A Task Planner for an Autonomous Social Robot. In *Distributed Autonomous Robotic Systems 8*, 335–344. Springer Berlin Heidelberg.

de Silva, L.; Pandey, A. K.; Gharbi, M.; and Alami, R. 2013. Towards combining HTN planning and geometric task planning. In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.

de Silva, L.; Gharbi, M.; Pandey, A. K.; and Alami, R. 2014. A new approach to combined symbolic-geometric

backtracking in the context of human-robot interaction. In *ICRA*, To Appear.

de Silva, L.; Pandey, A. K.; and Alami, R. 2013. An interface for interleaved symbolic-geometric planning and backtracking. In *IROS*, 232–239.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009a. Semantic Attachments for Domain-Independent Planning Systems. In *ICAPS*, 114–121.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009b. Integrating Symbolic and Geometric Planning for Mobile Manipulation. In *IEEE International Workshop on Safety, Security and Rescue Robotics*, 1–6.

Erol, K.; Hendler, J.; and Nau, D. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *International Conf. on AI Planning Systems*, 249–254.

Fleury, S.; Herrb, M.; and Chatila, R. 1997. Genom: A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture. In *IROS-97*, 842–848.

Fong, T.; Nourbakhsh, I. R.; and Dautenhahn, K. 2003. A survey of socially interactive robots. *Robotics and Autonomous Systems* 42(3-4):143–166.

Guitton, J.; Warnier, M.; and Alami, R. 2012. Belief Management for HRI Planning. In *Workshop on Belief change, Non-monotonic reasoning and Conflict resolution*.

Ingrand, F. F.; Chatila, R.; Alami, R.; and Robert, F. 1996. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *ICRA*, 43–49.

Karlsson, L.; Bidot, J.; Lagriffoul, F.; Saffiotti, A.; Hillenbrand, U.; and Schmidt, F. 2012. Combining task and path planning for a humanoid two-arm robotic system. In *Workshop on Combining Task and Motion Planning for Real-World Applications*, 114–122.

Klein, G.; Woods, D. D.; Bradshaw, J. M.; Hoffman, R. R.; and Feltovich, P. J. 2004. Ten Challenges for Making Automation a "Team Player" in Joint Human-Agent Activity. *IEEE Intelligent Systems* 19(6):91–95.

Lagriffoul, F.; Dimitrov, D.; Saffiotti, A.; and Karlsson, L. 2012. Constraint propagation on interval bounds for dealing with geometric backtracking. In *IROS*, 957–964.

Nau, D.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *IJCAI*, 968–973.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; and Wu, D. 2003. SHOP2: An HTN Planning System. In *JAIR*, volume 20, 379–404.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pandey, A. K.; Saut, J.-P.; Sidobre, D.; and Alami, R. 2012. Towards planning human-robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement. In *IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics*, 1371–1376.

Siméon, T.; Laumond, J.-P.; and Lamiraux, F. 2001. Move3D: a generic platform for path planning. In *4th International Symposium on Assembly and Task Planning*, 25–30.

Tate, A. 1976. Project Planning Using a Hierarchic Nonlinear Planner. In *Department of AI Research Report No. 25, University of Edinburgh*.

Warnier, M.; Guitton, J.; Lemaignan, S.; and Alami, R. 2012. When the robot puts itself in your shoes. Managing and exploiting human and robot beliefs. In *RO-MAN*, 948–954.

A Cooperative Model-based Control Agent for a Reconfigurable Manufacturing Plant

Stefano Borgo, Amedeo Cesta, Andrea Orlandini, Riccardo Rasconi, Marco Suriano
CNR – National Research Council of Italy
Institute for Cognitive Science and Technology
{name.surname}@istc.cnr.it

Alessandro Umbrico
Roma TRE University
Department of Engineering
alessandro.umbrico@uniroma3.it

Abstract

This paper presents an overview of the use of planning and execution techniques in nodes of a Reconfigurable Transportation Systems (RTSs). A manufacturing plant is here conceived as multiple independent modules to implement alternative inbound logistic systems' configurations. To support this capability of the robotic hardware, an integrated solution is proposed using timeline-based planning and control responsible for managing both the node regular activities and reconfiguration activities. A cooperation layer dedicated to multi-robot coordination completes the overall architecture.

Introduction

The capability of production environment to dynamically cope with changes of the production requirements is seen as one of the key enabling factors for highly automated and competitive production systems (Wiendahl et al. 2007; Terkaj, Tolio, and Valente 2009). In this regard, Reconfigurable Manufacturing Systems (RMSs), see e.g., (Crawford et al. 2013; Ruml, Do, and Fromherz 2005; Do, Ruml, and Zhou 2008), are endowed with a set of reconfigurability capabilities that can be related either to the single component of the system (e.g., a mechatronic device) or to the entire production cell and system layout. The role of such dynamic capabilities is to implement the correct system reconfiguration in response, for instance, to a change of the production demand. Such fast adaptation capabilities entail that the control architecture integrates knowledge-based and modular features, so as to allow for high reconfigurability from both a reasoning and mechanical standpoint.

This work describes the mid-term outcomes of the GECKO project. GECKO (Generic Evolutionary Control Knowledge-based mOdule) proposes an adaptive control infrastructure in which the production environment is modeled as a community of autonomous, self-declaring, collaborating GECKO modules encapsulated in the physical mechatronic equipment. Each GECKO node gathers the information from the environment, reproduces an abstraction of the shop-floor and interprets the production dynamics. This potentially allows evaluating, configuring and tuning the GECKO capabilities on the requirements by automatically activating the control functions and implementing resource production and energy efficiency optimization strategies over time.

To this aim, three main lines of research are active within the GECKO project: a distributed auction-based approach to part routing; a timeline-based control approach for supervisory node control; an ontology-based approach aiming at providing a viable connection between manufacturing concepts and control models.

This paper offers first an overview of the project then describes how P&S techniques are integrated with other features to serve the GECKO aims. The rest of the paper is subdivided in three parts: (1) a relevant case study in the manufacturing environment; (2) the comprehensive GECKO approach, and (3) the deployment of a state-of-the-art Planning & Execution system to address the project requirements on dynamic reconfiguration.

A Reconfigurable Shop-Floor Plant

The Pilot Case of the GECKO project is an automatic manufacturing system for printed circuit boards (PCB) reworking – see the birds eye diagram is presented in Figure 1. The objective of the system is to analyze defective PCBs, automatically diagnose their defects and, depending on the gravity of the malfunctions, attempt an automatic repair of the PCBs or send them directly to the shredding.

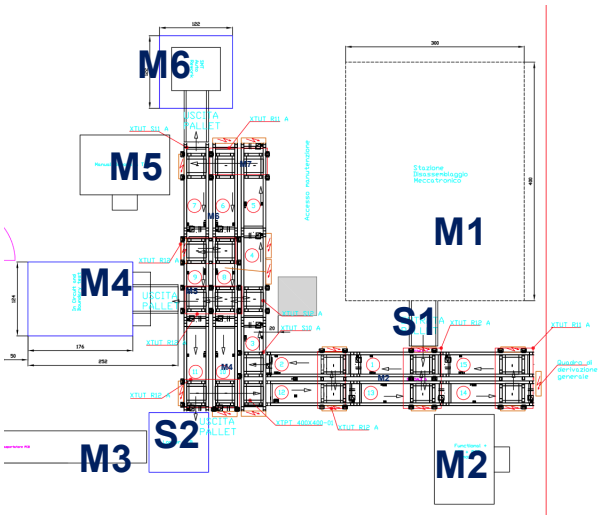
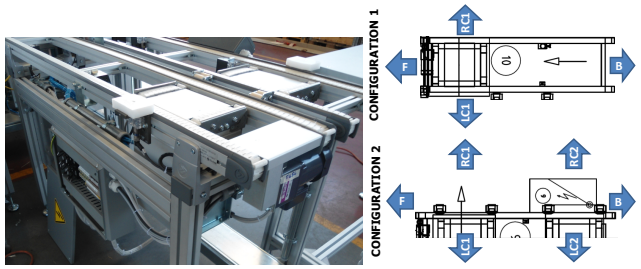


Figure 1: PCB Plant Shop Floor

More specifically, the system in Figure 1 contains 6 machines (M1, ..., M6) and a conveyor system that connects all

the machines. M1 is the loading/unloading cell, and represents both the entry point of all the parts (i.e., the PCBs) that have to be worked, and the exit point of all the PCBs that have undergone automatic successful repair. Machines M2 and M5 are manual repair stations, while machine M6 is an automatic repair station. Machine M4 is the in-circuit Diagnose&Testing station, and finally M3 is the shredding station, representing the exit point of all the PCBs either found damaged beyond repair at M2, or whose repair did not solve the malfunction.

All the stations are connected by means of a reconfigurable transportation system, composed of mechatronic components (i.e., transport modules), integrating dedicated sensors and actuators, as well as the related control system. Figure 2(a) provides a picture of a transport module. Each transport module is composed of three transportation units. The units belong to two major categories: unidirectional and bidirectional units; specifically the bidirectional units enable the lateral movement (i.e., cross-transfers) between two transportation modules. More specifically, each transport system module can support two main (straight) transfer services and one to many cross transfer services. Figure 2(b) depicts two possible configurations as an example.



(a) (b)

Figure 2: (a) A transport module; (b) Transport module transfer services

Configuration 1 supports the forward (F) and backward (B) transfer capabilities as well as a specific position (i.e., the bidirectional transportation unit) with left (LC1) and right (RC1) cross transfer capabilities. Configuration 2 extends Configuration 1 by the integration with a further bidirectional transportation unit (LC2 and RC2). The maximum number of bidirectional units within a module is limited just by its straight length, defined during the transport system module design phase, considering the number of pallet positions which are requested within a module (three, in this particular case).

The transport system modules can be connected back to back to form a desired conveyor layout, such as the one depicted in Figure 1, in a flexible and agile fashion. The PCBs manufacturing process requires PCB to be loaded on a fixturing system (pallet) in order to be transported to and processed by the machines. The transportation system is to move one or more pallets (i.e., a fixed number of pallets can simultaneously traverse the system) and each pallet can be either empty or loaded with a PCB to be processed. The transportation system plays two major tasks within the shop-floor: (i) the transportation of pallets with PCB to be processed by the system's machines and (ii) the transportation of pallets with PCB as an interoperation buffer. Figure 3 presents an example of routing configuration involving 9 transportation modules.

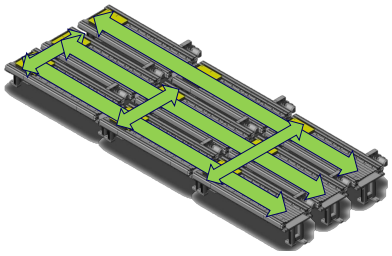


Figure 3: A possible routing configuration from 9 transportation modules

Such a flexible transportation system generally allows for a number of possible routing solutions for each single pallet which is associated to a given destination. Depending on the part type, and especially on the response given by the diagnosis machine M4 after the analysis of the current PCB, the pallet carrying the part will be assigned a different destination (e.g., the shredding station M3, or the loading/unloading cell M1), and this information will be made available only at execution time.

A GECKO Control Module

The GECKO production environment is composed by a community of control modules encapsulated in the physical mechatronic equipment that persistently communicate, cooperate and negotiate to guarantee the production goals. Each GECKO module operating in the shop-floor is equipped with a suitable I/O communication system capable of sensing the environment (i.e., data from sensors, other possibly connected entities, etc.) as well as publishing its own identity and capabilities, therefore providing a suitable information for identifying the production context and communicating its presence to the external world. Then, in the case study introduced above, every transportation module and every working machine composing the shop floor plant represents a GECKO module.

GECKO entities receive the information from the external environment and will interpret such information, thus inferring the actual production context (tasks and production goals). Consequently, they automatically recognize, negotiate and select, and configure the internal settings, functionalities and status. Their real-time controllers monitor the execution of the production processes and tasks with respect to both the dynamic status of the resources/modules of the transportation module and the general production context. After tasks completion, the entities communicate the successful execution to other entities, thus synchronizing themselves with the shop-floor environment.

To this aim, a layered control architecture has been designed to implement a cooperative model-based control agent for a reconfigurable manufacturing plant (Borgo et al. 2014). The architecture is defined as the composition of four interacting layers: a Coordination Layer, to manage inter-module communication and information exchange; a Production Layer, to implement a real-time controller to mon-

itor the execution of production tasks; a Control layer, to provide low-level functionalities; an Ontology-based layer, to enrich the module with knowledge-based capabilities.

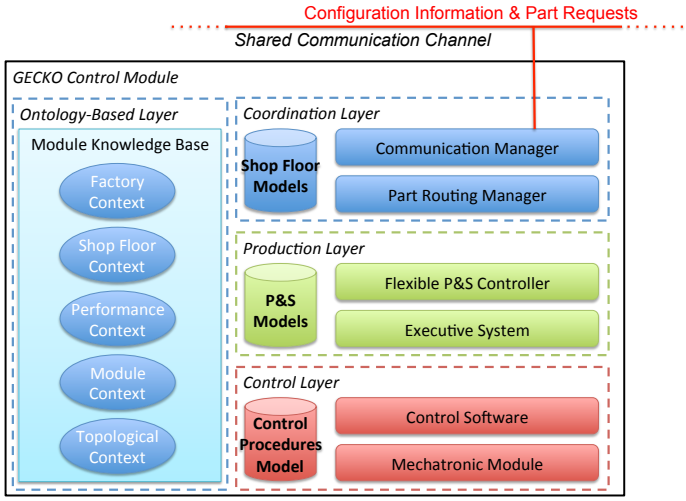


Figure 4: The GECKO Control Module Layered Architecture

Coordination Layer

One of the more crucial objective of the GECKO project is to realize a community of independent and self-contained modules able to cooperate together with the aim of managing the parts routing. In this regard, each module participates to a distributed reasoning process in order to dynamically define the work flow of the production environment. Then, a *Coordination Layer* has been designed in order to constitute the architectural element responsible to make the module able to interact with other modules as well all to participate in the distributed routing process.

The Coordination Layer is composed by two main components: a *Communication Manager*, in charge of providing an interface through a communication channel shared among all the GECKO control modules as well as implementing the module communication facility; a *Part Routing Manager*, responsible for negotiating with other modules the routing of working parts and pallets.

In (Carpanzano et al. 2014), a description of the specific work on the part routing dynamic policies is given. In particular, a distributed part routing approach inspired by auctioning techniques is proposed. That solution constitutes a reasoning system embedded in the GECKO modules in order to give them the autonomy to decide whether to take part in a certain transportation path and/or negotiate the routing policies, depending on the specific goals, while maintaining full awareness of the surrounding production environment.

With respect to the GECKO case study, the distributed reasoning process is started every time a new *pallet* enters the shop floor. The goal of the distributed process is to compute the pallet's path toward all the destinations according to the pallet's work-plan (i.e., the sequence of machine the part should visit to be properly worked). In this process every module participates in the definition of the path by making an evaluation of the feasibility of receiving that pallet. That is, every involved module computes a measure of the "suitability" of taking charge of the pallet and the path of the pallets is decided according to these values. The Coordination Layer implements such an assessment by reasoning on the Shop-Floor model and querying the production layer which provides timeline-based representation functionalities (see later) by means of which the module is able to reason about time and physical constraints.

Production Layer

The *Production Layer* is responsible for real-time control and to adapt the module's features to the production needs. It is responsible to make the module able to actually participate to the work flow of the shop floor, i.e., it interacts with the *Control Layer* by dispatching commands and gathering feedbacks in order to safely execute the planned activities. For this purpose it realizes the GECKO agent's control loop by integrating an APSI-TRF planner, called EPSL, integrated with a suitable executive system that will be further described in the next sections. The planner synthesizes the commands to dispatch at the *Control Layer* by reasoning on the set of high-level requests which is incrementally built by the *Coordination Layer* during the module life cycle. A more detailed description of the modules composing the Production Layer is provided later in the paper.

Control Layer

The *Control Layer* is the composition of a Control Software and a Mechatronic Module (i.e., either a Machine or a Transport Module). The GECKO control software is based on a distributed approach supported by an IEC61499 standard reference model. Each mechatronic module is then represented by a dedicated hardware resources virtually represented by a software function block encapsulating the module control logic. The GECKO control software has to be capable of dynamically changing their reactive behavior following the mechatronic system reconfigurations and provide low level failures detection. The automation services embedded within each software function block has to be activated/disactivated when a mechatronic module is integrated/removed from the automation system due to foreseen ad unforeseen events (e.g., production reconfigurations, system failures, maintenance). In order to support such online adaptation, dedicated software mechanisms have been designed and developed in a prototype version within the GECKO project. The description of such layer is out of the scope of the paper.

Ontology-based Layer

The GECKO project aims to develop an adaptive control infrastructure for manufacturing based on modular elements and distributed reasoning. A flexible and adaptable environment as the one foreseen in GECKO must be monitored and controlled within the system and by the system's elements via a shared and suitably rich language. Two aspects are particularly important in this perspective.

The first amounts to ensure that all the modules have access to the types of information relevant for their (actual and expected) functioning within the system. To this end, in GECKO the overall information system is based on the ontological analysis of the communication needs among the

modules. The aim of this analysis is to provide a coherent classification of data into information types and to ensure that the types cover the spectrum of relevant information.

The second aspect at the center of information management within the GECKO perspective is the adoption of an articulated information structure that allows each GECKO module to reliably interpret the information it receives and to coherently and consistently produce information to distribute.

In this regard, the idea of using ontology aims at defining a mechanism for dynamically generate a high-level description of the module by using a shared language. Thus, the *Ontology-based Layer* in Fig. 4 is responsible to use such a language to *build* the Knowledge Base which captures all the structural, performance and production information concerning the mechatronic device. Then, from the control perspective, the *Production Layer* is able to exploit Knowledge Base's information in order to dynamically infer the module actual status and build a model capturing all the information required to safely/effectively control the mechatronic module. Therefore, every time the Knowledge Base is updated, also the control model must be updated.

Then the GECKO Knowledge Base is to represent different types of information, like machine capacities, product classification and operational time for transportation, a clear and efficient classification of the information of interest cannot be built ad hoc without jeopardizing the flexibility and adaptability of the overall system. The adopted approach in facing such problem leads to disregard systems of classification based on roles (classification based on time constraints is particularly interesting in dealing with emergency situations but is too limited in standard situations) or on event evolution (classification by time is particularly interesting in dealing with unexpected changes or emergencies) and to focus first of all on the information content. Here, we briefly describe the GECKO approach to information modeling that introduces a global information structure based on discriminating contexts.

Knowledge Based and Contexts

The use of contexts in the GECKO framework is to filter information, as used at the shop floor level, according to a classification based on ontological principles. This means that contexts are exploited to *model* the factory along two perspectives: entity and data classification provided by the ontology on the one hand, and role of information provided by the contexts on the other. An analysis has been performed to identify relevant information flow and types of information that are potentially needed in a generic GECKO factory structure, and applied the methodology for ontology analysis used to develop DOLCE (Borgo and Masolo 2009). This led to identify types of information and how they can affect the reasoning and the activities of a given GECKO module.

Our analysis, which is carried out from the viewpoint of a GECKO module, suggested to separate three types of information: external, internal and topological. The 'external' contexts are about information the GECKO module cannot control nor modify. This amounts to three distinct subcontexts: the shared language in the system, the elements existing in the systems, and the system performance. The 'internal' context collects the information the GECKO element has about itself and its own capacities to act and change. Finally, the 'topological' context provides information on the relationship between the GECKO element at stake and its neighbour modules (and other agents), thus providing a local view of the topological setting across the agents.

External contexts. Since the GECKO modules are considered as constituting an integrated and coordinated shop floor, it is fundamental to ensure they understand and use a common language to exchange information. For this reason, one type of context is dedicated to the factory as a whole: Factory language context (FLC). This context is constituted by the language that all the modules of the factory must use for public communications. In particular the language provides vocabulary, rules and semantics to establish primarily functionalities and capacities that a modules can recognize and talk about; temporal and topological information that can be collected and shared, requests of actions and committed actions (shared plans).

A second type of external contexts collects information about the status of the factory at run-time: Factory Shop Floor context (FSC). This context is constituted by information on the elements (modules and products) presently at the shop floor and the requests for action or availability made or received by a GECKO module.

The last type of external contexts is related to the constraints and parameters that apply to the performance of the factory as a whole: Factory performance context (FPC). This context is constituted by information related to efficient use of the factory, productivity, energy consumption, throughput and other general constraints.

Internal context. In contrast to the external contexts, this type is dedicated to the local view of a single module. This context collects information which is about the internal structure and the capacities of the given module: Module internal context (MIC). This context is constituted by information about the single module (its identifier), its actual capacities (what it *can* possibly do) and, for each capacity, the time it takes to perform it plus auxiliary information (e.g., it can continuously perform action a for at most 5 min without break; it can perform the action on piece of min/max size/weight xyz etc.), the possible configurations and attributions it can undergo, how long it takes to actuate the change, possible limitations in the changes, timing for maintenance, when the next maintenance is scheduled, information on partial or total malfunctioning of some features and so on.

Topological context. Finally, each GECKO module is connected to other modules, its neighbors. The information about actual connections, coordination activities and commitments across neighbors (e.g., their agreed local schedules) is collected by the topological context: Module topological context (MTC).

In GECKO, we are currently generating a suitable ontology based on the set of contexts defined above and investigating ways for connecting such ontology and the timeline-based models the Production Layer uses to control a GECKO module. This is currently a challenging research goal we are pursuing and, here, we just give few examples of possible interesting connections.

For instance, the Module Internal Context describes the module internal composition, the set of transportation unit that logically compose the transportation module, their type, and module possible configurations, capabilities and timing information. This information can be exploited in order to define the set of state variables and components composing the planning domain and their possible states. Indeed, this information completely describes the module's physical composition together with its physical and temporal constraints. Similarly, the Functional Context describes the high-level functionalities the module exposes to other modules in the shop floor. These functionalities represent operational and/or maintenance activities (high-level requests) a module is able to perform during its lifecycle within the production environment. An high-level request can be decomposed in a set of actions the module has to perform in order to satisfy the request. Therefore this context describes dependencies and constraints that hold among module's components that guarantee a safe and successful completion of these activities. We can exploit this context to define the high-level *input state variable* whose values represent the requests received by the module over time. Moreover we can extract the set of rules and constraints that define the model *synchronization*s that allow the planner to safely synthesize the set of actions needed to satisfy the planning goals (i.e., the requests received by the planner).

A Flexible P&S Controller

To implement the Production Layer of a GECKO module, a timeline-based Planning and Scheduling (P&S) approach has been used aiming to design a flexible supervision module for GECKO. The timeline-based approach has been introduced in (Muscettola 1994) and has demonstrated successful in a number of space applications (Muscettola 1994; Jonsson et al. 2000; Cesta et al. 2007).

The modeling assumption underlying this approach is inspired by the classical Control Theory: the problem is modeled by identifying a set of relevant *components* whose temporal evolutions need to be controlled to obtain a desired behavior. Components are primitive entities for knowledge modeling, and represent logical or physical subsystems whose properties may vary in time. In this respect, the set of domain features under control are modeled as a set of temporal functions whose values have to be decided over a time horizon. Such functions are synthesized during problem solving by posting planning decisions. The evolution of a single temporal feature over a time horizon is called the *timeline* of that feature. In particular, for the purpose of this paper multi-valued *state variables* are considered as the basic type of time varying features (Muscettola 1994). As in classical control theory, the evolution of those features are described by some causal laws which determine legal temporal evolutions of timelines. For the state variables, such causal laws are encoded in a *Domain Theory* which determines the operational constraints of a given domain. Task of a planner is to find a sequence of control decisions that brings the variables into a final set of desired evolutions (i.e., the *Planning Goals*) always satisfying the domain specification.

APSI Timeline Representation Framework. The APSI-TRF is a software development framework for planning and scheduling, developed by our group within the Advanced Planning and Scheduling Initiative (APSI) promoted by the European Space Agency (ESA). The framework supports the development effort by providing a library of basic planning and scheduling domain independent solvers and a uniform representation of the solution database. It allows to represent several planning and scheduling concepts in the form of timelines. Indeed, components such as multi-valued state variables and resources like those commonly used in constraint-based schedulers provide enough modelling power for a good set of planning and scheduling needs.

It is worth noting that the APSI-TRF is not a planner per se but a development environment for planners and schedulers. Therefore to create a complete application it is necessary to insert a further module (a Problem Solver) on top of the domain representation in the Domain Layer. Such additional module is responsible for either driving a generic search or implementing a specific constructive heuristic for solving the problems at hand.

The EPSL planner. Aiming at the definition of a new P&S solver on top of the APSI-TRF, enhancing the definition of different general purpose solvers as well as preserving some interfaces with respect to the original tool, a research result achieved in GECKO was the definition of the Extensible Planning and Scheduling Library (Cesta, Orlandini, and Umbrico 2013) (EPSL).

The main goal of EPSL is to provide a planning environment in which it is possible to easily define new timeline-based planners, customized for the particular problem to address. The EPSL key point is the "planner-interpretation" which defines a timeline-based planner as the composition of several independent modules combined together according to a set of interfaces defined within the environment. EPSL, defines a planner as the tuple $\langle$P, S, H, E$\rangle$. P is the problem to solve. S is the strategy used to manage the search space fringe. The strategy S can be choosen among several built-in options (A*, DFS, BFS, etc). H is the heuristc exploited to analyze the current plan p and to extract the "more relevant" (according to H) flaw to be solved. A flaw represents either a goal or a threat that must be solved in order to find a valid solution plan. E is the resolver engine encapsulating the reasoning capabilities of the EPSL framework. E is composed by several solving algorithms, called resolvers, each of which is used to solve a particular type of flaw during any plan refinement.

The GECKO Planning Problem

The GECKO project represents an effort for developing a software module which is able to control a highly reconfigurable mechatronic device by dynamically adapting device's features to the specific needs of the production environment. To reach these challenging objectives, our idea is to combine the capabilities coming from two distinct worlds, the world of Ontology and the world of Planning. With Ontology we want to define a semantic for the information that describe device's features, e.g., device physical composition, device functionalities, etc. With Planning we want to exploit this structured information to automatically infer actual capabilities of the device and to dynamically control its features in

order to satisfy the production flow requests. As anticipated in the description of the reconfigurable shop-floor we are implementing, the GECKO project objective, w.r.t the case study described above, is to control a single *Transportation Module* (see Figure 2) for transporting pallets within a manufacturing plant (see Figure 1). The plant can be seen as a set of rails on which pallets are transported from one working machine to another. A *Transportation Module* is a component of these rails, it is responsible to actually move a pallet through the plant.

A *Transportation Module* (TM) can be seen as logically composed by one or more unit, called *Transportation Unit* (TU), that represent pallet "locations" inside a module during the transportation. A *Transportation Unit* can hold only one pallet at a time. The module is endowed with a main conveyor by means of which the module can transport pallets in FRONT and BACK directions (the module's base exchanging directions) and exchange them with other modules in the plant. In order to support additional exchanging directions, such as LEFT and RIGHT, some modules may be endowed with a special type of *Transportation Unit*, called *Cross Transfer*.

This *special* unit is endowed with an additional dedicated conveyor which allows the unit (and the module) to support additional transportation directions (i.e. LEFT and RIGHT). However a *Cross Transfer Unit* must be properly configured in order to transport the pallet toward the desired direction. A module has to switch to a proper configuration in order to transport a pallet towards FRONT or BACK direction, namely, it has to set all its *Cross Transfer Unit* in DOWN position in order to let them be able to use the main conveyor. Similarly, a module has to set a *Cross Transfer Unit* in UP position in order to transport a pallet toward an additional direction (i.e., LEFT or RIGHT). A *Cross Transfer Unit* can use its dedicated conveyor only in UP position.

The types of *Transportation Unit* composing a *Transportation Module* determine both the set of possible working configuration of the module and the module's functionalities. Let us take as example a module composed only by "simple" *Transportation Unit*s (i.e. without *Cross Transfer Unit*s). Such a module will be able to transport pallets by means of its main conveyor only (i.e., FRONT and BACK transportation directions only supported). Conversely, a module with one *Cross Transfer Unit* at least, will be able to transport pallets towards LEFT and RIGHT directions also, if properly configured.

A manufacturing plant can be composed by different types of modules where each module can have a different type and number of *Transportation Unit*s. In this context, we want to dynamically recognize the actual capabilities and functionalities of a module in order to produce a general description of the module (an abstract model). The planner deliberates on this model in order to adapt module's features to the production flow. When a module receives a request for transporting a pallet, the planner, given a consistent description of the module, is responsible to synthesize and execute the sequence of actions that allow to successfully satisfy the request.

Figure 5 illustrates the abstract representation of a generic *Transportation Module* we have taken into account as a case study. Thus, according to the above description of a module

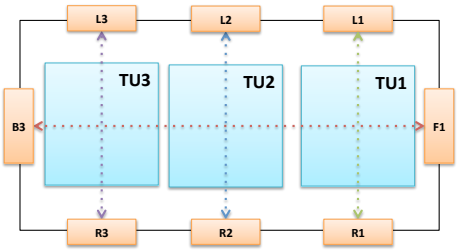


Figure 5: The GECKO Module Case Study

in the manufacturing plant of Figure 1, it is possible to identify the following elements: A main conveyor which allows to move pallets forward and backward through the module, the dashed red line in Figure 5; A set of *port*s {F1, B3, L1, L2, ..., R1, R2, ...} that allow the module to exchange pallets externally with other modules, the orange blocks in Figure 5; A set of *Transportation Unit* {TU1, TU2, TU3, ...} the module has to synchronize in order to safely transport pallets. In Figure 5 all TUs composing the module are *Cross Transfer Unit*s

A Timeline specification for the GECKO planning problem. The reasoning process of the planner relies on an abstract representation of the physical module to control which is represented by means of a timeline-based model. The model has to capture all (and only) the constraints and features of the system that are relevant to the control perspective. Therefore, according to the abstraction given in Figure

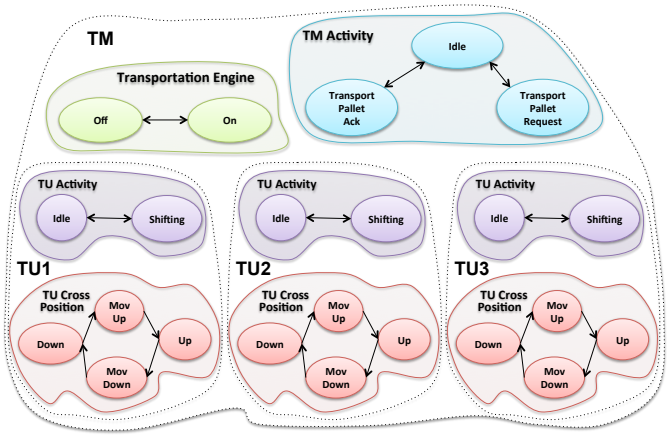


Figure 6: Transportation Module State Variables

5 the TM supports the following transportation directions {F1, B3, L1, L2, L3, R1, R2, R3} where the number near the direction label represents the Transportation Unit associated. It is possible to identify several *State Variable*s defining the physical constraints of the module, see Figure 6.

The Transportation Module Activity State Variable models the transportation requests (high-level goals) received by the module over time. The blue Automata depicted in Figure 6 defines the transition constraints among State Variable values: *Idle()*, no request to process; *TransportPalletRe-*

quest(?pid, ?in, ?out), the module receives a request to transport the pallet *?pid* entering the module from port *?in*, towards port *?out*. The ports combination identifies the transportation direction; *TransportPalletAck(?pid, ?in, ?out)*, the modules has satisfied a transportation request previously received. *The Transportation Unit Activity State Variable* models the possible states of a TU within the module lifecycle. The idea is to model a single TU like an *atomic* TM, i.e. a TM without TUs. The violet Automata depicted in Figure 6 defines the transition constraints among State Variable values: *Idle()*, no pallets are traversing the transportation unit; *Shifting(?pid, ?in, ?out)*, a pallet *?pid* is traversing the transportation unit from *?in* to *?out*. This state is equivalent to *Transport Pallet Request(?pid, ?from, ?out)* state of a TM. It represents that a pallet is traversing the TU from *?in* to *?out*. The parameters *?in* and *?out* defines *loca port*s that "logically" allow TUs exchanging pallets. *The Transportation Unit Cross Position State Variable* models the configuration of the transportation unit's *cross-transfer* engine. The red Automata of Figure 6 defines the transition constraints among State Variable allowed values: *Down()*, the unti's *cross-transfer* is set to be in down position; *MovingUp()*, the unit's *cross-transfer* is changing its configuration from down to up; *Up()*, the unit's *cross-transfer* is set to be in up position; *MovingDown()*, the unit's *cross-transfer* is changing its configuration from up to down. *The Transportation Engine State Variable* models the module's transportation engine status and its movement directions (its configurations). The green Automata depicted in Figure 6 defines the transition constraints among State Variable allowed values: *Off()*, the conveyor is not moving *On(?dir),* the conveyor is moving toward direction *?dir*. The parameter *?dir* identify the transportation direction of a pallet.

State Variables define constraints guiding the timeline building process. They describe the set of values the state variable can assumes, i.e. values which can be on the component's timeline, and the set of state transitions allowed, i.e. the value sequences admitted on the component's timeline (*legal behaviour*s). However, they do not describe how these components work together.

State Variables describe the component types of the transportation module and the constraints that should be satisfied in order to correctly use a single instance of these component types. Further constraints are needed in order to coordinate the components and to plan a pallet request (i.e., satisfy a *goal*). Thus, when a module receives a request, it has to suitably synchronize all its components (i.e. its *Transportation Unit*s and conveyors) in order to satisfy such requested operation. And to model such information, additional constraints, called *synchronization*s are needed to define temporal constraints among timeline's values in the form of *Allen's Interval algebra* (Allen 1983).

A synchronization applies to a component's decision in order to specify conditions that must hold to avoid inconsistencies with respect to other component behaviors. Figure 7 illustrates the timeline-based plan resulting from a request for transporting a pallet from F1 to B3 (w.r.t. the model in Figure 5). The dashed arrows represent temporal constraints among values introduced applying *synchronization*s in order to guarantee a consistent behaviour of the overall module. Such *synchronization*s are to guarantee that functional

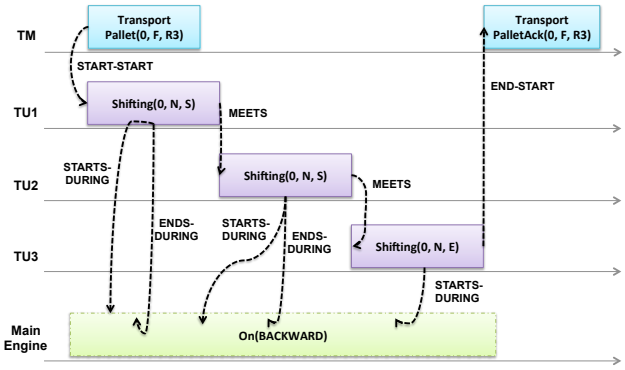


Figure 7: A Timeline-based plan for a GECKO module

requirements on one or more timelines (the *target* of the *synchronization*) hold when the "origin" timeline assumes a particular value (the *reference* value of the *synchronization*).

The constraints depicted in Figure 7 belong to the same *synchronization* and define the values module's component must assume during pallet traversal. When a module receives a request for transporting a pallet from F1 to B3, the *Transportation Unit* - TU1 - must be set in *Shifting()* state, it means that the unit is ready to receive the pallet from nearby module and send the pallet to TU2.

So, as soon as *Shifting()* state ends on TU1, another *Shifting()* state starts on TU2 and the same happens between TU2 and TU3. This means that when a pallet is traversing from a TUx to a TUy, both must be ready to respectively send and receive the pallet. Every time a *Shifting()* state occurs on a TU, the associated temporal interval must starts and ends during the *On()* state of the module's main conveyor. That is, while TUs are exchanging pallets the main conveyor must be turned on in order to actually move the pallet through the module.

Moreover, looking again at Fig. 7, it is possible to notice that the value *On()* of the main conveyor's timeline has a dashed border. It means that the pallet traversal may require more than one *On()* value, i.e. the pallet may stay without moving on one or more TUs during the traversal. Indeed, every *Shifting()* interval may starts and/or ends during different *On()* intervals. Therefore a *Shifting()* interval duration is not fixed in time. The number of *On()* values needed to satisfy the request depends on other activities planned by the module, i.e. it depends on the set of constraints (synchronizations) the planner has to consider during the planning process.

The reasoning capabilities needed to synthesize such a plan are obtained by integrating in the control infrastructure a timeline-based planner developed within the EPSL library. This library provides a set of general purpose *operator*s that allow to build timeline-based plans. Relying on these functionalities, it is possible to easily define timeline-based planners that can be tuned according to the specific features of the problem to address. Therefore we defined an *ad-hoc* GECKO timeline-based planner we used to dynamically control and adapt modules to the production environment.

A GECKO Executive Module

Previous works have tackled the robust execution issue within a Constraint-based Temporal Planning (CBTP) framework deploying specialized techniques based on temporal-constraint networks. Several authors (e.g., (Morris and Muscettola 2005; Hunsberger 2010)) have proposed a *dispatchable execution* approach where a flexible temporal plan is then used by a plan executive that schedules activities on-line while guaranteeing constraint satisfaction. This general line of research has concerned specifically the use of timeline-based planning and their temporal constraint networks implementation for an homogeneous synthesis of controllers. Among the architectures that use a uniform representation for the continuous planning and execution task are IDEA (Muscettola et al. 2002), T-REX (Py, Rajan, and McGann 2010) and, more recently, GOAC (Ceballos et al. 2011).

Within the GECKO project, the research effort has been focused on a complete deploy of our current effort on robust plan execution (Orlandini et al. 2013). In particular, we have extended the timeline-based, domain independent deliberative control system, called APSI *Deliberative Reactor* (ADR) described in (Fratini et al. 2011). This section summarizes the recent advances on the enhanced ADR system. The ADR has been designed to address a set of open issues in planning and execution with timelines, i.e., the dynamic management of goals during planning and execution, the assessment of the status of partially executed goals and the dynamic dispatching of commands. More in detail, the ADR is an instance of a proactive control system entirely based on APSI-TRF technology and is constituted by (i) an execution module, to dispatch planned timelines, to supervise their execution status and to entail continuous planning and re-planning, (ii) a timeline-based planning module, i.e., EPSL, to model and solve planning problems.

The ADR is designed to be domain independent, i.e., once provided with a suitable timeline-based description model of the system to be controlled and a set of temporal goals to be achieved it fully implements all the required functionalities to plan for goals, dispatch planned values to the controlled system and supervise plan execution collecting the telemetry of the controlled system. One of the main advantage of domain independence is the capability of the deliberative reactor to both plan for user goals and dynamically react to off-nominal conditions detected from the controlled system telemetry. Additionally, it allows flexibility in two direction: it can achieve different classes of user goals in the same system by substituting the controller model and it can be deployed to control different systems by substituting the domain description of the controlled system.

Within GECKO, it has been investigated the integration in APSI-TRF of an alternative and novel approach to flexible plan dispatching/execution proposed in (Orlandini et al. 2011), where robust plan execution is pursued by relying on Timed Game Automata (TGA) formal modeling and controller synthesis. The technique used to synthesize plan controllers is a direct consequence of the formalization proposed in (Cesta et al. 2010) in which plan correctness as well as dynamic controllability are checked by means of TGA model checking. Analogously to that work, the dynamic P&S domain and the generated flexible temporal plan are encoded into TGA models. However, a different perspective is exploited through the use of a model checker (i.e., UPPAAL-TIGA (Behrmann et al. 2007)) to directly synthesize a real-time plan controller for the flexible plan. Such controller guarantees robust plan execution along with dynamic controllability.

In (Orlandini et al. 2013), an experimental evaluation of the TGA-based method has been reported discussing the practical feasibility of the on-line deployment of such TGA-based approach in different operative modalities and considering increasingly complex instances of a real-world robotics case study. The reported results show the viability of the approach as well as confirm the benefits of two important general advantages: i) the presented methodology relies on off-the-shelf planning/verification tools and, thus, it enables its application to any generic layered control architecture that integrates a temporal P&S system; ii) the possibility of applying different settings for the control system allows to look for trade-off between planning, verification and execution costs, i.e., the control system can be tuned up according to the actual criticality of the controlled system.

TGA-based controllers for flexible plan execution. *Timed Game Automata* (Maler, Pnueli, and Sifakis 1995) (TGA) allow to model real-time systems and controllability problems representing uncontrollable activities as *adversary moves* within a game between the controller and the environment. Following the same approach presented in (Cesta et al. 2010), flexible timeline-based plan verification can be performed by solving a *Reachability Game* using UPPAAL-TIGA. To this end, flexible timeline-based plans, state variables, and domain theory descriptions are compiled into a network of TGA (nTGA). This is obtained by means of through following steps: (1) a flexible timeline-based plan $\mathcal{P}$ is mapped into a nTGA *Plan*. Each timeline is encoded as a sequence of locations (one for each timed interval), while transition guards and location invariants are defined according to (respectively) lower and upper bounds of flexible timed intervals; (2) the associated set of state variables *SV* is mapped into a nTGA *StateVar*. Basically, a one-to-one mapping is defined between state variables descriptions and TGA. In such encoding, value transitions are partitioned into controllable and uncontrollable according to their actual execution profile; (3) an *Observer* automaton is introduced to check for violations of both value constraints and Domain Theory. In particular, two locations are defined: an Error location, to state constraint violations, and a Nominal (OK) location, to state that the plan behavior is correct. The *Observer* is defined as fully uncontrollable. (4) the compound nTGA $\mathcal{PL} = \textit{StateVar} \cup \textit{Plan} \cup \{\textit{Observer}\}$ encapsulates flexible plan, state variables and domain theory descriptions.

Then, considering a Reachability Game $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) where *Init* represents the set of the initial locations of each automaton in $\mathcal{PL}$, *Safe* is the OK location of the Observer automaton, and *Goal* is the set of goal locations (one for each automaton in *Plan*), plan verification can be performed solving/winning the $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) defined above. In order to win/solve the reachability game RG, UPPAAL-TIGA is exploited as verification tool checking a suitable CTL formula, i.e., $\Phi = A$ *[Safe U Goal]* in $\mathcal{PL}$. In fact, the formula Φ states that along all its possible temporal evolutions, $\mathcal{PL}$ remains in *Safe* states until *Goal*

states are reached. That is, in all the possible temporal evolutions of the timeline-based plan $\mathcal{P}$ all the constraints are fulfilled and the plan is completed. Thus, if the solver verifies the above property, then the flexible temporal plan is valid. Whenever the flexible plan is not verified, UPPAAL-TIGA produces an execution trace showing one temporal evolution that leads to a fault. Such a strategy can be analyzed in order to check either for plan weaknesses or for the presence of flaws in the planning model.

Furthermore, a mapping between flexible temporal behaviors defined by $\mathcal{P}$ over the temporal horizon $[0, H]$ and the automata behaviors defined by $\mathcal{PL}$ can be shown: for each partial temporal behavior $pb \in \mathcal{P}$ defined over $H' < H$, it there exists a unique temporal evolution ρ_{pb} of $\mathcal{PL}$ such that ρ_{pb} represents the partial temporal behavior pb over the same horizon H'. That is, ρ_{pb} represents the same valued intervals sequence in $\mathcal{P}$ limited to H' and the duration of ρ_{pb} is exactly the horizon H'. As a consequence, the winning strategy f generated by UPPAAL-TIGA solving the reachability game on $\mathcal{PL}$ represents a flexible plan controller $\mathcal{C}_f$ that achieves the planning goals maintaining the dynamic controllability during the overall plan execution. In (Orlandini et al. 2011), the reader finds a formal account of the generation of a plan controller $\mathcal{C}_f$ derived from a winning strategy f generated by UPPAAL-TIGA.

Integrating the TGA-based controller in the ADR. Here, the integration in the ADR of the TGA-based method discussed above is presented. In particular, a suitable embedding of the UPPAAL-TIGA tool within the ADR planning and execution cycle is shown and, then, the advantages in terms of plan correctness and robust execution enforcement (i.e., dynamic controllability) are discussed.

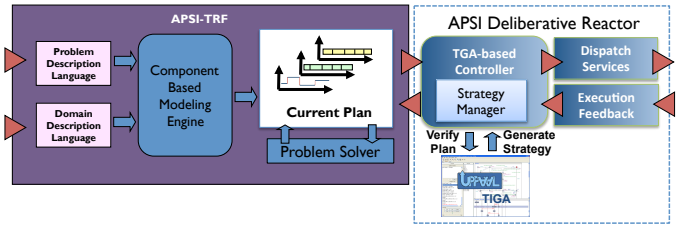


Figure 8: *Integration of TGA-based controller in the* APSI *Deliberative Reactor*

The integration schema is shown in Figure 8. The left part of the figure shows the APSI-TRF general architecture. The domain and problem models are encoded as Domain Definition Language (DDL) and Problem Definition Language (PDL) input files. Then, both DDL and PDL files are parsed and managed by the *Component-based Domain Modeling Engine* and a *Current Plan* (i.e., the initial planning problem) is created to be manipulated by a *Problem Solver*. Indeed, the Current Plan is specialized as a data structure called Decision Network in APSI-TRF. Then, a generic problem solver, e.g., EPSL, applies a solving procedure until the Current Plan satisfies all the planning goals (or fails in finding a solution plan).

The right part of Fig. 8 depicts a simplified view of the APSI Deliberative Reactor with two relevant services, i.e., the *Dispatch services* and the *Execution Feedback* modules, in charge of (respectively) dispatching suitable commands for the controlled system and collecting feedback from the field. The new APSI Deliberative Reactor architecture still reflects the structure of a T-REX reactor (as defined in (Fratini et al. 2011)) as well as it introduces two new components, i.e., the *TGA-based Controller* (TC) and the *Strategy Manager* (SM), enabling robust plans execution through the use of strategies generated by UPPAAL-TIGA.

The TC is in charge of managing plans in order to (i) verify plan correctness and (ii) generate a dynamically controllable execution strategy: once a solution plan $\mathcal{P}$ is generated by the problem solver (i.e., the stored Current Plan is actually the valid plan to be executed), the TC automatically generates the associated TGA encoding ($\mathcal{PL}$) and, then, invokes UPPAAL-TIGA in order to verify the correctness of the plan as well as to check for the existence of (at last) one temporal plan execution guaranteeing the correct achievement of the plan goals, independently from the exogenous events generated by the environment (i.e., enforcing the dynamic controllability). If the verifier finds one of these sequences, then a strategy for the plan execution is generated. Namely, a strategy generated by UPPAAL-TIGA is a set of *temporal rules* that should guide the controlled system through the *execution space* avoiding plan failures during its execution. More formally, an UPPAAL-TIGA strategy is a set of rules $f(t, s)$ defined as follows:

$$f(t,s) = \begin{cases} t_{wl} < t < t_{wu} & \text{Wait} \\ t_{al} < t < t_{au} & \text{Action } a_n \\ t > t_{err} & \text{Error} \end{cases}$$

where t is the execution time, s is one of the possible state of the system, t_{wu} and t_{wl} represent, respectively, lower and upper bounds of a time interval in which the system must wait for the environment to act, t_{al} and t_{au} represent lower and upper bounds of a time interval in which the system should perform the action a_n (i.e., one of the timeline should change value) and t_{err} is a time limit beyond which the system generates an error. The latter represents the case in which the execution strategy is coping with exogenous events that are not properly modeled in the planning domain, e.g., the actual duration of an uncontrollable event is shorter/longer than the minimal/maximal duration stated in the domain model. This implies that the planning model is inconsistent with the actual behavior of the controlled system and, thus, a revision of that model (and the TGA encoding) is required.

The SM is the module in charge of implementing the concrete dispatching policy relying on the UPPAAL-TIGA strategy. In fact, once generated, the SM exploit such strategy to to choose the more suitable $f(t, s)$ rule to be executed, thus, extracting the associated action to be dispatched (or to *wait* while the controlled system is evolving) as well as to continuously monitor the internal status of the reactor timelines and the execution feedback received from the field.

Given the above, the new integrated reactor architecture guarantees plans correctness as well as the robust execution of the generated plans, thus, increasing the probability of successfully performing the temporal plan.

Conclusions

This paper has described the GECKO control module offered as a proposal for a Reconfigurable Transportation System. The general GECKO architecture has been synthesized during the first year of the project and the single components have been developed and tested under laboratory conditions. During the second year, recently started, an integration of all the components will be provided in the pilot plant for re-manufacturing of electronic components. Then, an extensive experimental tests will be performed to assess the robustness and effectiveness of the comprehensive approach. Among future works, the dynamic management of unforeseen events (such as, e.g., maintenance tasks, module failures and transportation delays) by means of the integration of suitable mechanisms for robust replanning and/or plan repair is currently under investigation.

Acknowledgments. CNR authors are supported by MIUR/CNR Flagship Initiative FdF-SP1-T2.1 Project GECKO "Generic Evolutionary Control Knowledge-based mOdule". Thanks to the colleagues of ITIA and IEIIT for the collaboration in the project. A special thanks to Anna Valente and Emanuele Carpanzano for the long term collaboration on the integration of AI in Reconfigurable Manufacturing.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26(11):832–843.

Behrmann, G.; Cougnard, A.; David, A.; Fleury, E.; Larsen, K.; and Lime, D. 2007. UPPAAL-TIGA: Time for playing games! In *Proc. of CAV-07*, number 4590 in LNCS, 121–125. Springer.

Borgo, S., and Masolo, C. 2009. Foundational choices in DOLCE. In Staab, S., and Studer, R., eds., *Handbook on Ontologies*, International Handbooks on Information Systems. Springer Berlin Heidelberg. 361–381.

Borgo, S.; Cesta, A.; Orlandini, A.; Rasconi, R.; Suriano, M.; and Umbrico, A. 2014. Towards a cooperative knowledge-based control agent for a reconfigurable manufacturing plant. In *ETFA-2014. Proc. 19th IEEE Int. Conf. on Emerging Technologies and Factory Automation*.

Carpanzano, E.; Cesta, A.; Orlandini, A.; Rasconi, R.; and Valente, A. 2014. Intelligent dynamic part routing policies in plug&produce reconfigurable transportation systems. *CIRP Annals – Manufacturing Technology* (in press).

Ceballos, A.; Bensalem, S.; Cesta, A.; de Silva, L.; Fratini, S.; Ingrand, F.; Ocon, J.; Orlandini, A.; Py, F.; Rajan, K.; Rasconi, R.; and van Winnendael, M. 2011. A Goal-Oriented Autonomous Controller for Space Exploration. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Policella, N. 2007. An Innovative Product for Space Mission Planning: An A Posteriori Evaluation. In *ICAPS-07*, 57–64.

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2010. Analyzing Flexible Timeline Plan. In *ECAI 2010. Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215. IOS Press.

Cesta, A.; Orlandini, A.; and Umbrico, A. 2013. Toward a general purpose software environment for timeline-based planning. In *20th RCRA International Workshop on "Experimental Evaluation of Algorithms for solving problems with combinatorial explosion"*.

Crawford, L. S.; Do, M. B.; Ruml, W.; Hindi, H.; Eldershaw, C.; Zhou, R.; Kuhn, L.; Fromherz, M. P.; Biegelsen, D.; de Kleer, J.; et al. 2013. Online reconfigurable machines. *AI Magazine* 34(3).

Do, M. B.; Ruml, W.; and Zhou, R. 2008. On-line planning and scheduling: An application to controlling modular printers. In *AAAI*, 1519–1523.

Fratini, A.; Cesta, A.; De Benedictis, R.; Orlandini, A.; and Rasconi, R. 2011. APSI-Based Deliberation in Goal Oriented Autonomous Controllers. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

Hunsberger, L. 2010. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *Temporal Representation and Reasoning (TIME), 2010 17th International Symposium on*, 121–128.

Jonsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proceedings of the Fifth Int. Conf. on AI Planning and Scheduling*.

Maler, O.; Pnueli, A.; and Sifakis, J. 1995. On the Synthesis of Discrete Controllers for Timed Systems. In *STACS*, LNCS, 229–242. Springer.

Morris, P. H., and Muscettola, N. 2005. Temporal Dynamic Controllability Revisited. In *Proc. of AAAI 2005*, 1193–1198.

Muscettola, N.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proc. of NASA Workshop on Planning and Scheduling for Space*.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Orlandini, A.; Finzi, A.; Cesta, A.; and Fratini, S. 2011. Tga-based controllers for flexible plan execution. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI.*, volume 7006 of *Lecture Notes in Computer Science*, 233–245. Springer.

Orlandini, A.; Suriano, M.; Cesta, A.; and Finzi, A. 2013. Controller synthesis for safety critical planning. In *IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*, 306–313. IEEE.

Py, F.; Rajan, K.; and McGann, C. 2010. A Systematic Agent Framework for Situated Autonomous Systems. In *AAMAS-10. Proc. of the 9^{th} Int. Conf. on Autonomous Agents and Multiagent Systems*.

Ruml, W.; Do, M. B.; and Fromherz, M. P. 2005. On-line planning and scheduling for high-speed manufacturing. In *ICAPS-05. Proc. 15th Int. Conf. on Automated Planning and Scheduling*, 30–39.

Terkaj, W.; Tolio, T.; and Valente, A. 2009. A review on manufacturing flexibility. In *Design of Flexible Production Systems*. Springer. 41–61.

Wiendahl, H.-P.; ElMaraghy, H. A.; Nyhuis, P.; Zäh, M. F.; Wiendahl, H.-H.; Duffie, N.; and Brieke, M. 2007. Changeable manufacturing-classification, design and operation. *CIRP Annals – Manufacturing Technology* 56(2):783–809.

Continual Planning via Reconfiguration and Goal Revision

Enrico Scala
Dipartimento di Informatica
Corso Svizzera 185
Torino - Italy
scala@di.unito.it

Abstract

The execution of plans in real world domains can be several times threatened by the occurrence of unexpected contingencies. Beside propositional conditions, realistic scenarios require plans to satisfy constraints on consumable resources, too. Relying on the notion of execution modalities, the paper presents a continual planning system for the efficient management of unforeseen resources consumption. In particular, the proposal combines plan reconfiguration and goal revision in order to efficiently (i) recovery from the impasse and (ii) reasoning on the current mission constraints whenever the reconfiguration is not possible. Both mechanisms exploit a CSP encoding and have been experimentally validated on a planetary rover domain. Experiments analyze the benefits (and the limits) of the approach in terms of competence and efficiency under different timeout settings.

Introduction

The plan execution in real world domains is a critical activity that has to take into account several challenges (Ghallab, Nau, and Traverso 2014). In particular, in realistic scenarios, an agent operates in an environment which is just partially observable and loosely predictable; unexpected contingencies can arise at each step of the execution. As a consequence, the agent must have some form of autonomy in order to guarantee robust plan execution (i.e., accomplishing the mission).

Robust plan execution has been tackled in two ways: on-line and off-line. On-line approaches, such as (Gerevini and Serina 2010; van der Krogt and de Weerdt 2005; Garrido, C., and Onaindia 2010; Brenner and Nebel 2009; Micalizio 2013), interleave execution and replanning: whenever some unexpected contingency makes the plan unfeasible, the plan execution is stopped and a new plan is synthesized as a result of a new planning phase. Off-line approaches, such as (Block, Wehowsky, and Williams 2006; Conrad and Williams 2011), avoid replanning by anticipating, at planning time, possible contingencies. The result of such a planning phase is a contingent plan that encodes choices between functionally equivalent sub-plans[1]. At execution time, the plan executor is able to select a contingent plan according to the current contextual conditions. However, as for instance in the work of (Policella et al. 2009), the focus is mainly on the temporal dimension and they do not consider consumable and continuous resources.

This paper presents an on-line methodology to deal with unexpected deviations in the resources consumption. First, in line with the action-based approach *a-la* STRIPS (Fox and Long 2003) and differently from the constrained based planning (Fratini, Pecora, and Cesta 2008; Muscettola 1993), we model consumable resources as numeric fluents (introduced in PDDL 2.1 (Fox and Long 2003)). Then, we enrich the model of the agent's actions by expliciting a set of *execution modalities*. The basic idea is that the propositional effects of an action can be achieved under different configurations of the agent's devices. These configurations, however, may have a different impact on the consumption of the resources. An *execution modality* abstracts the low level behavior of an action modeling its resource consumption profile when such an action is carried out in a given configuration. The integration of *execution modality* at the PDDL level allows a seamless integration between planning and execution.

Relying on the concept of execution modalities, we handle exceptions as a reconfiguration of action modalities, rather than as a replanning problem (Scala, Micalizio, and Torasso 2014). In particular, the approach at the basis of this paper is a plan execution strategy, denoted as ReCon; once (significant) deviations from the nominal trajectory are detected, ReCon intervenes by reconfiguring the modalities of the actions still to be performed with the purpose of restoring the validity of resource constraints imposed by the agent mission. As we will see the reconfiguration could be handled by means of a CSP encoding.

Whenever the adaptation turns out to be unfeasible, as a novel contribution, the paper combines the reconfiguration with a goal reasoning facility. Exploiting the CSP interpretation above, and recent advancement in the SAT literature (Marques-Silva et al. 2013; Liffiton and Sakallah 2008), the paper shows how extending a SAT based methodology for the goal reasoning problem. In particular, our approach aims at individuating the set(s) of goals preventing the CSP to find a valid allocation of modalities. Doing so, the agent (or the human operator supervising its execution) can have a more accurate description of the problem and (if possible) can de-

[1]The notion of alternative (sub)plans is also present for (off-line) scheduling; for details see (Barták, Ĉepek, and Hejna 2008)

cide to relax one or more constraints in order to make the reconfiguration newly feasible.

In extension to the results presented in (Scala, Micalizio, and Torasso 2014), this paper provides new experimental evidences of the benefit of the reconfiguration characterization over replanning. In particular, we tested the reconfiguration for near real-time settings, that is where deliberative activities have to be performed very quickly, in order to be useful for the acting.

For reasoning on the arising CSP formulation, the paper uses Choco[2] as solver, for both the reconfiguration and the goal reasoning task. Doing so, the approach turns out to be solver independent.

After introducing a motivating example, the paper describes the employed action model, enriched with the notion of execution modality. Then we introduce the ReCon strategy and the goal revision mechanism. Afterwards, an example shows how the system actually works in a exploration rover mission. The paper concludes with an experimental section evaluating the competence and the efficiency of the strategies reported in this paper. In particular, the reconfiguration is experimentally compared with the LPG-ADAPT system (Gerevini, Saetti, and Serina 2012).

Motivating Example

Let us consider a planetary rover in charge of exploring (and analyzing) a number of potentially interesting sites and able to transmit information towards the Earth. In doing so the rover is capable of moving, taking pictures, and starting the data upload once the pieces of information must be transmitted. For simplicity reasons, consider the mission plan of Figure 1, involving *take picture, drive* and *communications* activities. This mission represents a feasible solution for a planning problem with goal: {`in(r1,l3), mem>=120, pwr>=0, time<=115`} ; that is, at the end of plan the rover must be located in `l3` (propositional fluent), the free memory must be (at least) 120 memory units, there must be a positive amount of power, and the mission must be completed within 115 secs.

The figure shows how the four actions (regular boxes) change the status of the rover over the time (rounded-corner boxes)[3]. Note that the status of a rover involves both propositional fluents, (e.g., `in(r1, l1)` means that the rover `r1` is in location `l1`); and numeric fluents: `memory` represents the amount of free memory, `power` is the amount of available power, `time` is the mission time given in seconds, and `com_cost` is an overall cost associated with communications.

The estimates about the rover's status are inferred by predicting, deterministically, the effects of the actions. In particular, the numeric fluents have been estimated by using a "default setting" (i.e., a standard modality) associated with each action.

[2]The software is at disposal at http://www.emn.fr/z-info/choco-solver/, while the work has been presented in (Narendra, Rochart, and Lorca 2008)

[3]To simplify the picture, we show in the rover's status just a subset of the whole status variables

Let us now assume that during the execution of the first drive action the rover has to travel across a rough terrain. Such an unexpected condition affects the drive as the rover is forced to slowdown[4], and as a consequence the drive action will take a longer time to be completed; the effects are propagated till the last snapshot, *s_4* where the goal constraint `time <= 115` will be no longer satisfied.

After detecting this inconsistency, approaches based on a pure replanning step would compute a new plan achieving the goal by changing the original mission. For instance, some actions could be skipped in order to compensate the time lost during the first drive.

However, robotic systems as a planetary rover have typically different configurations of actions to be executed and each configuration can have a different impact on the mission progress. For instance the robotic systems described in (Calisi et al. 2008) and in (Micalizio, Scala, and Torasso 2011) can perform a drive action in fast or slow modes. Reliable transmission to the earth, for example, can be slow and cheap, or fast and expensive, depending on the devices actually used.

Our proposal is to explicitly represent such different configurations within the action models, and hence try to resolve an impasse via a reconfiguration of the actions still to be performed. Intuitively, our objective is to keep the high level plan structure unchanged, but to adjust the modalities of the actions still to be performed.

In the next section we will introduce the action model adopted that explicitly expresses the set of *execution modality* at disposal.

The Action Model

This section introduces the action model adopted in this work. The model exploits (and extends) the numeric PDDL 2.1 action model (Fox and Long 2003), i.e. where the notion of numeric fluents has been proposed. In particular, we use the numeric fluents to model continuous and consumable resources.

The intuition is that, while actions differ each other in terms of qualitative effects (e.g. a drive action models how the position of the rover changes after the action application), the expected result of an action can actually be obtained in many different ways, for instance by appropriately configuring the rover's devices (e.g. the drive action can be performed with several engine configurations). Of course, different configurations have in general different resource profiles and it is therefore possible that the execution of an action in a given configuration would lead to a constraint violation (or it is not applicable in that setting), whereas the same action performed in another configuration would not. We call these alternative configurations *modalities* and we propose to capture the impact of a specific modality by modeling the use of specific configurations in terms of pre/post conditions on the numeric fluents involved; such *modalities* become explicit in the action model definition.

[4]The slowdown command of the rover may be the consequence of a reactive supervisor, which operates as a continuous controller as shown in (Micalizio, Scala, and Torasso 2011)

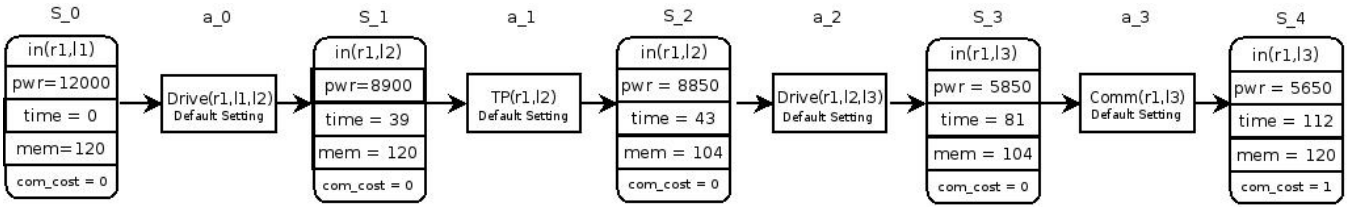


Figure 1: A simple mission plan.

The resulting model expresses the actions at two different levels of abstraction. The higher one is the qualitative level indicating "what" the action does. The lower one is the quantitative level expressing "how" the action achieves its effect.

The idea of *alternative behaviors* has also been investigated in (off-line) scheduling, where the notion of Temporal Network with Alternatives has been introduced (Barták, Ĉepek, and Hejna 2008). It is quite evident however that, as anticipated in the introduction, the concept of execution modality is inspired to an (on-line) action centered approach (Brenner and Nebel 2009), rather than on a constraints/scheduling based one (Cesta and Fratini 2009).

By recalling our motivating example, Figure 2 shows the model of the drive action. The action template `drive (?r, ?l1, ?l2)` requires a rover `?r` to move from a location `?l1` to location `?l2`. `:modalities` introduces the set of modalities associated with a `drive`; in particular, we express for this action, three alternative modalities:
- `safe`: the rover moves slowly and far from obstacles; intuitively the action should spend more time but consuming less power
- `cruise`: the rover moves at its cruise speed and can go closer to obstacles;
- `agile`: the rover moves faster than `cruise`, consuming more power but requiring less time.

The `:precondition` and `:effect` fields list the applicability conditions and the effects, respectively, and are structured as follows: first a propositional formula encodes the condition under which the action can be applied; the second field (`:effect`) indicates the positive and the negative effects of the action. For each modality *m* in `:modalities` we have the amount of resources required (numeric precondition) or consumed/produced (numeric effect) by the action when performed under that specific modality *m*.

For instance, the preconditions `(reachable ?l1, ?l2)` and `(in ?r1, ?l1)` are two atoms required as preconditions for the application of the action. These two atoms must be satisfied independently of the modality actually used to perform the `drive` action. While the comparison `(safe: (>= (power ?r) (* (safe_cons ?r) (/ (distance ?l1 ?l2) (safe_speed ?r)))))` means that the modality `safe` can be selected when the rover's power is at least larger than a threshold given by evaluating the expression on the right side. Analogously, `(safe: (decrease (power ?r) (*(safe_cons ?r) (/ (distance ?l1 ?l2) (safe_speed ?r))))` describes in the effects how the rover's power is reduced after the execution of the drive action. More precisely, we have modeled the power consumption as a function depending on the duration of the drive action (computed considering distance and speed) and the average power consumption per time unit given a specific modality. For instance, in safe modality, the amount of power consumed depends on two parameters `(safe_cons ?r)` and `(safe_speed ?r)` which are the average consumption and the average speed for the safe modality, respectively, while `(distance ?l1 ?l2)` is the distance between the two locations `?l1` and `?l2`.
Finally, note that in the numeric effects of each modality, the model updates also the fluent `time` according to the selected modality. Also in this case, the duration of the action is estimated by a function associated with each possible action modality.

Analogously to the drive action we model modalities also for the Take Picture (TP) and the Communication (COMM). For TP we have the low (LR) and high (HR) resolution modalities which differ in the quality of the taken picture and the occupied memory. Intuitively, the more the resolution is, the more the memory consumption will be. Whereas for the Communication we assume to have two different channels of transmissions: CH1 with low overall `comm_cost` and low bandwidth, and CH2 with high overall `comm_cost` but high bandwidth.

The selection of action modalities has to take into account that complex dependencies among resources could exist. For instance, even if a high resolution TP takes the same time as a low resolution TP, the selection has a big impact on the amount of time spent globally, too. As a matter of facts, as long as the amount of stored information increases, the time spent by a (possible) successive COMM grows up accordingly, which means that also the global mission horizon will be revised.

Given the actions defined so far, a plan is a total ordered set of fully instantiated action templates[5]. Given a state S and a set of goals G to be reached (including both propositional/classical conditions and constraints on the amount of resources, (Fox and Long 2003)), the mission plan is valid iff it achieves G from S, and each action is executable in the trajectory of states resulting from the plan application.

Executing the mission plan. As we have seen in the previous section, the plan can be threatened many times by unex-

[5]The plan can be also generated automatically by exploiting a numeric planner system, properly modified to handle actions with modalities. (e.g., the Metric-FF planning system (Hoffmann 2003) or LPG (Gerevini, Saetti, and Serina 2008)

```
(:action drive
  :parameters ( ?r - robot ?l1 - site ?l2 - site)
  :modalities (safe,normal,agile)
  :precondition (and (in ?r ?l1) (road ?l1 ?l2)
  (safe: (>= (power ?r) (* (safe_cons ?r)
     (/ (distance ?l1 ?l2) (safe_speed ?r)))))
  (cruise: (>= (power ?r) (* (cruise_cons ?r)
            (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
  (agile: (>= (power ?r) (* (agile_cons ?r)
            (/ (distance ?l1 ?l2) (agile_speed ?r)))))
 )
  :effect
  (and
    (in ?r ?l2) (not (in ?r ?l1))
    (safe: (decrease  (power ?r) (* (safe_cons ?r)
           (/ (distance ?l1 ?l2) (safe_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (safe_speed ?r)))
           (increase  (powerC ?r) (* (safe_cons ?r)
                      (/ (distance ?l1 ?l2) (safe_speed ?r))))
    (cruise: (decrease  (power ?r) (* (cruise_cons ?r)
                      (/ (distance ?l1 ?l2) (cruise_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (cruise_speed ?r))
           (increase  (powerC ?r) (* (cruise_cons ?r)
                      (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
    (agile: (decrease  (power ?r) (* (agile_cons ?r)
                      (/ (distance ?l1 ?l2) (agile_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (agile_speed ?r))
           (increase  (powerC ?r) (* (agile_cons ?r)
                      (/ (distance ?l1 ?l2) (agile_speed ?r)))))
 )
```

Figure 2: The augmented model of a `drive` action.

pected contingencies; so the validity of the mission can be easily compromised during its actual execution.

Nevertheless, when the detected unexpected contingency invalidates the resource consumption, a replanning mechanism could be an excessive reaction. While the current modality allocation would not be valid with the constraints involved in the plan and in the goal, there could be "other" allocations of modalities still feasible. By exploiting this intuition, the next section introduces the adaptive execution technique which, instead of abandoning the mission being executed, tries first to repair the flaws via a reconfiguration of the action modalities. The reconfiguration considers all those actions still to be executed.

As we will see moreover, even in case of failure, i.e. there are no valid reconfigurations, by exploiting this characterization, the system is able to provide the user with an explanation of the occurred impasse indicating a set of possible constraints that should be relaxed in order to make the mission feasible.

Given a plan P, to indicate when a plan is just *resource invalid*, we will use the predicate *res_invalid* over P, i.e. we will say *res_invalid(P)*. Otherwise we will say that the plan is valid or structurally invalid. This latter case happens when, given the current plan formulation, at least an action in the plan is not propositional applicable, or there is at least a missing (propositional) goal. Moreover, we will say that the plan is consistent when there is at least an allocation of modalities such to satisfy the goal. Otherwise we will say that the plan is not consistent. As we will see, this last definition highlight the relation between the plan consistency and the underlying CSP encoding.

The Continual Planning Strategy

As anticipated in the introduction, for the successful execution of the plan, it should be supervised all along the task execution. Algorithm 1 shows the main steps required to accomplish this task. The algorithm takes in input the initial state S_0, the mission goal *Goal*, and the plan *P* expressed as discussed in the previous section. Note that each action has to have a particular modality of execution instantiated. The algorithm returns *Success* when the execution of the whole mission plan achieves the goal; *Failure* otherwise.

In this case, a failure means that there is no way to adapt the current plan in order to reach the goal satisfying mission constraints. To recover from this failure, a replanning step altering the structure of the plan should be invoked.

The first step of the algorithm is to build a *CSPModel* representing the mission plan (line 1). Due to lack of space, we cannot present this step in details; our approach, however, inherits the main steps by Lopez et al. in (Lopez and Bacchus 2003) in which the planning problem is addressed as a CSP[6]. As a difference w.r.t. the classical planning, the encoding exploited by our approach needs to store variables for the modalities to be chosen, and variables for the numeric fluents involved in the plan. Numeric fluents variables are replicated as many steps in the plan. The purpose is to capture all the possible evolutions of resources profiles given the modalities that will be selected. The constraints oblige the selection of the modality to be consistent with the resource belonging to the previous and successive time step. Moreover, further constraints allow only reconfigurations consistent with the current observation acquired (which at start-up corresponds to the initial state), and the goals/requirement of the mission.

Once the *CSPModel* has been built, the algorithm loops over the execution of the plan. Each iteration corresponds to the execution of the *i*-th action in the plan. At the end of the action execution the process verifies the current observation obs_{i+1} with the rest of the mission to be executed. In case the plan is structurally invalid (some propositional conditions are not satisfied or the goal cannot be reached) ReCon stops the plan execution and returns a failure; i.e., a replanning procedure is required.

Otherwise we can have two other situations. First, there have been no consistent deviations from the nominal predictions therefore the execution can proceed with the remaining part of the plan. Second the plan is just resource invalid ($res\_incon(P)$, line 10). In this latter case, ReCon has to adapt the current plan by finding an alternative assignments to action modalities that satisfies the numeric constraints (line 11). If the adaptation has success, a new non-empty plan *newP* is returned and substituted to the old one. This new plan is actually the old plan, but with a different allocations of action modalities. Otherwise, the plan cannot be adapted and a failure is returned; in this case, the plan execution is stopped and a new planning phase (or a revision of the current goal constraints, see next section) is needed.

Update

The **Update** step is sketched in Algorithm 2. The algorithm takes in input the CSP model to update, the last performed

[6]Alternative CSP conversions are possible; for instance see (Barták and Toropila 2010)

Algorithm 1: ReCon

```
   Input: S_0, Goal, P
   Output: Success or Failure
1  CSPModel = Init(S_0, Goal, P) ;
2  i = 0;
3  while ¬ P is completed do
4      execute(a_i, curMod(a_i));
5      obs_{i+1} = observe();
6      if P is structurally invalid w.r.t. obs_{i+1} and Goal
       then
7          return Failure
8      else
9          Update(CSPModel,a_i,num(obs_{i+1}));
10         if res_incon(P) then
11             newP = Adapt(CSPModel,i,Goal,P);
12             if newP ≠ ∅ then
13                 P = newP
14             else
15                 return Failure
16     i = i + 1
17 return Success
```

action a_i, and the set *NObs* of observations about numeric fluents. The algorithm starts by asserting within the model that the i-th action has been performed; see lines 1 and 2 in which variable mod_i is constrained to assume the special value *exec*. In particular, a first role of the *exec* value is to prevent the adaptation process to change the modality of an action that has already been performed. *exec* allows the acquisition of observations even when the observed values are completely unexpected. In fact, by assigning the modality of action a_i to *exec*, we relax all the constraints over the numeric variables at $(i+1)$-th step (which encode the action effects). This is done in lines 3-5 in which we iterate over the numeric fluents N^j mentioned in the effects of action a_i, and assign to the corresponding variable at $(i+1)$-th step the value observed in *NObs*. On the other hand, all the numeric fluents that are not mentioned in the effects of action a_i do not change, so the corresponding variables at step $i+1$ assume the same values as in the previous i-th step (lines 6-8). The idea of the Update is to make the CSP aware of the current new observations and the modalities already executed. In this way, a reconfiguration task does not need to rebuild the structure completely from scratch.

Adapt

The **Adapt** procedure, shown in Algorithm 3, takes in input the CSP model, the index i of the last action performed by the rover, the mission goal, and the plan P; the algorithm returns a new adapted plan, if it exists, or an empty plan when no solution exists.

The algorithm starts by removing from *CSPModel* the constraints on the modalities of actions still to be performed; i.e., each variable mod_k with k greater than i is no longer constrained (a_i is the last performed action and its modality is

Algorithm 2: Update

```
   Input: CSPModel, a_i,NObs
   Output: modified CSPModel
1  delConstraint(CSPModel,mod_i=curMod(a_i));
2  addConstraint(CSPModel,mod_i=exec);
3  foreach N^j ∈ affected(a_i) do
4      addConstraint(CSPModel,
5        (mod_i=exec)→ N^j_{i+1}=get(NObs,N^j_{i+1}))
6  foreach N^j ∈ ¬affected(a_i) do
7      addConstraint(CSPModel,
8        (mod_i=exec)→ N^j_{i+1}=N^j_i)
```

set to *exec*) (lines 1-2). This step is essential since the current modalities allocation inside the *CSPModel* is not valid; that is, the current assignment of modalities does not satisfy the global or the local constraints. By removing these constraints, we allow the CSP solver to search in the space of possible assignments to modality variables (i.e., the actual decisional variables, since the numeric fluents are just side effects of the modality selection), and find an alternative assignment that satisfies the global constraints (line 3). If the solver returns an empty solution, then there is no way to adapt the current plan and **Adapt** returns no solution. Otherwise (lines 6-10), at least a solution has been found. In this last case, a new assignment of modalities to the variables mod_k $(k : i+1..|P|)$ is extracted from the solution, and this assignment is returned to the ReCon algorithm as a new plan *newP* such that the actions are the same as in P, but the modality labels associated with the actions $a_{i+1},..,a_{|P|}$ are different.

Note that, in order to keep updated the CSP model for future adaptations, the returned assignment of modalities is also asserted in *CSPModel*; see lines 6 to 10.

Algorithm 3: Adapt

```
   Input: CSPModel, i,Goal,P
   Output: a new plan, if any
1  for k=i+1 to |P| do
2      delConstraint(CSPModel mod_k=currentMod(a_k))
3  Solution = solve(CSPModel);
4  if Solution = null then
5      return ∅
6  else
7      newP=extractModalitiesVar(Solution);
8      for k=i+1 to |newP| do
9          addConstraint(CSPModel,
           mod_i=curMod(newP[i]))
10     return newP
```

Goal Revision

In the previous section we have seen that when the observations invalidate the current configuration, an adaptation in-

side the modalities search space is started. Of course this step is sufficient when the underlying CSP formulation still contains solutions, whereas it is ineffective when there is no valid allocation at disposal; that is, the flexibility provided by the reconfiguration is not enough.

While this flexibility could be obtained by means of a generative approach, e.g. by replanning completely from scratch, if the problem is over-constrained, neither a replanning mechanism could be able to recover the mission.

There could be situations where the current mission constraints are too strict, while a small relaxation (for some constraints) could be beneficial for newly achieving consistency (i.e., at least a configuration of action modalities). The problem is hence to understand whether it is actually the case.

By reasoning directly on the (i) CSP interpretation provided by the encoding presented in previous section, (ii) the model of the actions, and (iii) the problem we are interested in, we consider constraints split into two different sets[7]:

- System constraints
- User constraints

System constraints are the ones generated by the action model (and in particular by the model of execution modalities), and the current observation coming from the environment; while the user constraints are the ones expressed in the goals set (which are the numeric constraints a given reconfiguration has to satisfy).

From a constraints perspective, we hence are interested in answering to the following question: which is the minimum set of user constraints such that if we remove these constraints from the CSP, then the CSP becomes newly consistent?

For instance, in our mission rover example it could be the case that the discrepancy between the predicted time and the actual time spent by the rover is actually very large, and also a reconfiguration is unable to recovery the situation. For this reason a possible way to restore the CSP consistency could be to relax the time constraint. However, it is interesting to note that also the constraint on the power could be relaxed, as a new allocation for the modalities can be found by promoting all the actions to be performed quicker.

In the context of diagnosis, satisfiability and constraint programming, a very similar issue is referred as the problem of computing the MCS (Minimal Correcting Set) of an inconsistent SAT/CSP. Recently this topic has received a quite big attention (Marques-Silva et al. 2013). In that work, in particular, multiple invocations of the solver are employed in order to recognize the threatening constraint(s).

In our scenario, to limit the computational impact due to the combinatorial explosion of cases to consider (which relies on considering a CSP consistency problem for each element of the power-set of the constraints involved in the goal), we focus just on those sets whose cardinality is minimal. These are in fact the most interesting ones in our scenario, as they represent just the less impact on the user preferences.

[7] A similar characterization is also usual in the context of model based diagnosis; for instance see (Felfernig, Schubert, and Reiterer 2013)

Similarly to the work presented by (Marques-Silva et al. 2013), our approach exploits the idea to reformulate the CSP in a partial MAXCSP, whose objective is to maximize the satisfiability of the constraints in the goal set, while preserving the consistency of the ones belonging to the system constraints set. The solution of this problem makes evident which are the satisfiable constraints, and hence, by reasoning on the complementary set, the ones threatening the CSP consistency.

Unfortunately, in the general case, there could be several sets of constraints of this kind, each of them with minimum cardinality; for this reason the mechanism has to be repeated as many of them can be inferred. To exploit the same MAXCSP mechanism, the intuition is to keep trace to the ones already discovered.

Algorithm 4 reports the relative pseudo-code.

Algorithm 4: MCS

Input: C : System Constraints, G : User Constraints, k : Integer
Output: S : Minimal Correcting Sets

```
1 S = ∅
2 while true do
3     S' = CO-MAXCSP(C,G,S)
4     if |S'| ≤ k then
5         S = S ∪ {S'}
6         k = |S'|
7     else
8         return S
```

The procedure takes in input the system constraints (C), the user constraints (G) and an integer k.

C and G indicate the sets of constraints as described above, while k aims at limiting the search to sets of cardinality less or equal to k.

The algorithm exploits the same CSP solver employed for the reconfiguration task, but in a different modality. More precisely the invocation to the solver is due to CO-MAXCSP routine. This function finds (incrementally) all the set of constraints which should be removed for achieving consistency of the CSP. The third parameter allows to keep trace of the constraints previously selected; such constraints are formulated as hard constraints in the CSP representation. At the end of the process, each element s from S will be such that $s \subseteq G$.

This mechanism completes the tools at disposal to our agent. Let us see now how the mission rover example can be handled by exploiting these facilities.

Running the Mission Rover Example

Let us consider again the example in Figure 1, and let us see how ReCon manages its execution. First of all, the plan model must be enriched with the execution modalities as previously explained; Figure 3 (top) shows the initial configuration of action modalities: the drive actions have `cruise` modalities, the take picture (TP) has `HR` (high resolution)

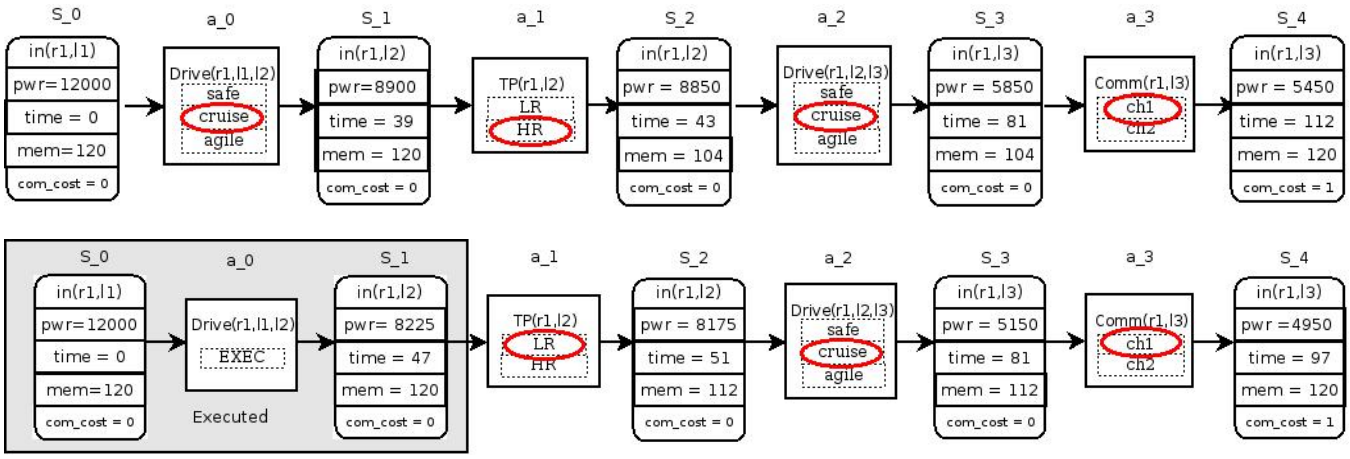


Figure 3: The initial configuration of modalities (above), and the reconfigured plan (below).

modality, and the communication (`Comm`) uses the low bandwidth channel (`CH1`). This is the enriched plan ReCon receives in input.

Now, let us assume that the actual execution of the first drive action takes a longer time than expected, 47s instead of 38s, and consumes more power, 3775 Joule instead of 3100 Joule. While the discrepancy on power is not a big issue as it will not cause a failure, the discrepancy on time will cause the violation of the constraint `time <=115`; in fact, performing the subsequent actions in their initial modalities would require 120 seconds. In other words, the assignment of modalities to the subsequent actions does not satisfies the mission constraints. This situation is detected by ReCon that intervenes and, by means of the **Adapt** algorithm discussed above, tries to find an alternative configuration of modalities.

Let us assume that communication cost is constrained; that is, the mission goal includes the constraint `com_cost = 1`; this prevents ReCon from using the fast communication channel. The more intuitive decision is to promote the execution of the *drive* to *agile*. However, this would cause the violation on the constraint concerning the maximum amount of power to be spent. Therefore ReCon has to look for an alternative assignments of modalities.

Observing the model of the action, it is interesting to note that a lower resolution image consumes less memory, meaning that the successive communication, in our case *(COMM R1 L3)*, will need less time (and also less power) for achieving its effects. For this reason ReCon demotes the next activity, i.e. TP, to be executed to modality LR and so the global constraints are now satisfied.

Of course, we assume that mission constraints leave ReCon some room to repair resource inconsistent situations. For instance, if the mission has required an hard constraint on the quality of the taken images, the low resolution would have not been possible, and hence an overall replanning would have been necessary.

In principle, by flattening all the actions and the given modalities as explained in (Scala 2013b), replanning is possible as alternative to the reconfiguration mechanism. In this case, however, the problem to be handled would become much more difficult, since all the possible action sequences applicable starting from the current state could be explored.

To highlight the complexity arising from a replanning formulation, let us assume that in our example there is a connection from location l3 to l4, and from l2 and l4. That is, the rover can move not only from l1 to l2, but also from l2 to l4 and from l4 to l3, for *all* the provided modalities. In addition, for simplicity reasons, assume that from that point (l3), the only possible sequence of actions toward the goal is given by a_2 and a_3.

While the reconfiguration mechanism can focus just on the impact on resources given by the selection of modalities for the next actions (tp, drive, comm), it is quite evident that a traditional replanner should deal with a larger search space. As matter of fact, it should consider also the (several) possible trajectories of states given by exploring the alternatives ways of reaching location l2 (drive(r1,l2,l4)), for all the possible modalities of execution. That is, it will have to cope with both the propositional and resource constraints of the arising planning problem. For a deeper discussion on this aspect, see (Scala 2013b).

To limit the intervention of the replanning mechanism also in this situation, or in case the plan structure should not be violated for any reasons (e.g., in space exploration scenarios), the goal revision could be activated. In this case the mechanism will find that a solution is possible whenever either the constraint on the info-loss, or the communication or the time should be removed or relaxed. This information could be provided to the ground control station that will decide the preferred solution, for instance by reasoning on the constraint that requires the less relaxation.

Experimental Validation

To assess the effectiveness of our proposal, we evaluated three main aspects: (1) the efficiency (time spent for the deliberation throughout the plan execution process), (2) the competence (the ability of completing a mission) and (3) the goal revision mechanism.

For the sake of evaluation we compared ReCon with a state of the art mechanism for plan repair, i.e. LPG-ADAPT[8]. In our experimental setup we used the same high level continual planning mechanism developed around ReCon. That is, whenever the plan becomes invalid from the point of view of the resources, LPG-ADAPT stops the execution of the plan and try to recover from the impasse.

We have implemented ReCon in Java 1.7 by exploiting the PPMaJaL library[9]; the Choco CSP solver (version 2.1.3) [10] has been used in the **Adapt** algorithm to find an alternative configuration. LPG-ADAPT is used by converting the rover actions with modalities in PDDL 2.1 actions. In order to emulate an (on-line) plan execution context, we stressed the ability of the system to provide solutions in a near real time way. In particular our experiments have been performed by considering 4 different timeout configurations (1, 2, 5 and 10 seconds). We aim at understanding the behavior of the reconfiguration mechanism in extreme conditions.

Our tests set consists of 168 plans; each plan involves up to 34 actions (i.e., drives, take pictures, and communications), it is fully instantiated (a modality has been assigned to each action), and feasible since all the goal constraints are satisfied when the plan is submitted for the execution.

To simulate unexpected deviations in the consumption of the resources, we have run[11] each test in four different settings. In each of these settings we have noised the amount of resources consumed by the actions. In particular, in setting 1, an action consumes 10% more than expected at planning time. In setting 2, the noise was increased to 20%, and so on until in setting 4 where the noise was set to 40%, i.e. an action consumes 40% more resources than initially predicted.

Figure 4 reports the competence - measured as the percentage of cases accomplished with success - for the two strategies, in the four settings of noise we have considered, over all the timeout configurations. As expected, the competence decreases as long as the amount of noise increases, for all the strategies tested. ReCon turned out to be more competent than LPG-ADAPT in the first two timeout settings. Even though LPG-ADAPT can modify all the aspects of the plan structure, and hence it is theoretically more competent than ReCon, the search spaces generated by the overall arising planning problems turned out to be too large from the point of view of LPG-ADAPT. As a consequence it often trespassed the time limit. We can observe a significant gap in the 1 second setup, while the margin is limited in the 2 secs setting. In the 10 secs setting LPG-ADAPT is instead more competent as it is able to exploit the flexibility provided by the overall numeric planning problem. As expected, the gap between ReCon and LPG-ADAPT decreases as long as the noise increases; this is of course due to the contribution of the flexibility of the search space in which LPG-ADAPT can find a solution.

	10%	20%	30%	40%
UNSOL(%)	100	100	100	100
CPU-TIME(msec)	154	98	233	103
GOALACH(%)	0	29	82	85

Table 2: Goal Revision Strategy Performances. Percentage of unsuccessful cases where ReCon has been able to shows the un-solvability (UNSOL), the average time spent for inferring the set of constraints to be relaxed (CPU-TIME), the percentage of cases where the goal revision strategy provided the user with at least a constraint to be relaxed.

Table 1 reports the cpu-time score for the two systems. For providing an informative parameter we have adopted the International Planning Competition metric[12]. That is, each case submitted is evaluated according to $\frac{T^*}{T}$, where T^* and T are the time spent by the best and the evaluated system, respectively. A not solved case takes 0.

For this aspect, the advantage of ReCon is very large. In fact, even for the worst case (when the noise is set to be 40%), ReCon is extremely efficient; in our raw data we measured that it takes, on average, just 356 msec. The score of course decreases as long as the contribution of the coverage comes into play.

Finally, in order to evaluate the feasibility of the goal revision strategy, we performed a preliminary testing phase on the same set of cases. In particular, we focused our attention on cases where the reconfiguration mechanism was not able to provide a solution. Table 2 collects the percentage of situations where the system was able to verify the un-solvability of the reconfiguration problem (UNSOL), the average time spent for inferring the candidate goals to relax (in order to achieve CSP consistency) (CPU-TIME), the percentage of situations where the goal revision strategy has been able to find a relaxation (GOALACH). In these experiments we set our parameter k to 1, so the system can relax at maximum one constraint in the goal set.

The system turned out to be very efficient, and for high levels of noise also quite informative. As matter of facts, when the noise is large, the system was able to provide a goal revision strategy for the most of the situations (84%, 85%). In particular the strategy was able to indicate which of the numeric goal constraints should be relaxed in order to make possible an alternative actions modality configuration.

It is important to note that, for the unsuccessful cases,

[8] LPG-ADAPT, (Gerevini, Saetti, and Serina 2012), is the plan adaptation extension of LPG, (Gerevini, Saetti, and Serina 2008), one of the more awarded systems throughout the planning competitions of the last decade. In this experimental phase we used the "speed" parameter, which substantially improves the performance of the system.

[9] www.di.unito.it\~ scala

[10] The Choco Solver implements the state of the art algorithms for constraint programming and has already been used in space applications, see (Cesta and Fratini 2009). Choco can be downloaded at http://www.emn.fr/z-info/choco-solver/.

[11] Experiments have run on a 2.53GHz Intel(R) Core(TM)2 Duo processor with 4 GB.

[12] As it has been noticed in occasion to the various International Planning Competitions, the average CPU time is few informative as planners tend to consume the time at disposal in a very different way from a case to another case; for further details see http://ipc.icaps-conference.org/

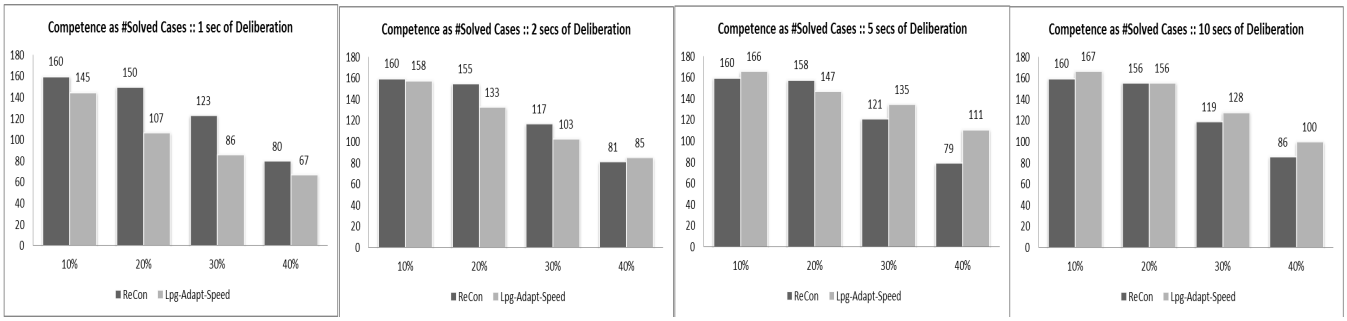


Figure 4: Competence of the ReCon and LPG-ADAPT. Each bar represents the number of cases that have been solved by a system, for each noise, over all the timeout configurations.

	1sec		2secs		5secs		10secs	
	ReCon	**LPGADAPT**	**ReCon**	**LPGADAPT**	**ReCon**	**LPGADAPT**	**ReCon**	**LPGADAPT**
10%	**155**	69	**156**	73	**155**	73	**154**	73
20%	**143**	68	**145**	87	**155**	59	**145**	92
30%	**117**	62	**109**	74	**143**	58	**109**	88
40%	**78**	52	**76**	71	**106**	67	**80**	76
total	**493**	251	**486**	305	**559**	257	**488**	329

Table 1: Cpu-Time score according to the International PLanning Competition Metric, for the two systems, over all the noise settings and timeout configurations.

while the reconfiguration task has been always able to show the absence of alternative configurations, the complexity of the numeric planning task completely prevented the LPG-ADAPT to terminate the execution with a response. As matter of facts, apart for very few cases, the unsuccessful situation reported in the architecture with the LPG-ADAPT have been due to the occurrence of the timeout.

Conclusion

We have proposed in this paper an approach for dealing with plan execution in presence of consumable resources. Rather than recovering from plan failures via a re-planning step (see e.g., (Gerevini and Serina 2010; van der Krogt and de Weerdt 2005; Garrido, C., and Onaindia 2010; Scala 2013a)), the work presented in this paper relies on a methodology, called ReCon, based on the re-configuration of the plan actions. ReCon is justified in all those scenarios where a pure replanning approach is unfeasible. This is the case, for instance, of a planetary rover performing a space exploration mission. Albeit a rover must exhibit some form of autonomy, its autonomy is often bounded by two main factors: (1) the on-board computational power is not always sufficient to handle mission recovery problems, and (2) the rover cannot in general deviate from the given mission plan without the approval from the ground control station.

ReCon presents advantages w.r.t. generative (e.g., LPG-ADAPT) approaches. First of all, as the experiments have demonstrated, reconfiguring plan actions is computationally cheaper than synthesizing a new plan by adaptation (as the one reported in (Gerevini, Saetti, and Serina 2012)). Moreover, ReCon leaves the high-level structure of the plan (i.e., the sequence of mission tasks) unchanged, but endows the agent with an appropriate level of autonomy for handling unexpected contingencies. In particular, this paper presented a novel experimental analysis showing that, also if we constrain the search with very strict timeout settings, ReCon continues to be very effective.

As a methodological innovation with respect to a previous work on this topic (Scala, Micalizio, and Torasso 2014), relying on the reconfiguration characterization, the paper proposed a novel strategy for reasoning about the numeric goal constraints to be achieved at reconfiguration level. This strategy can be used for instance in a mixed initiative setting to make the user aware about the causes of the reconfiguration inconsistency, and surely it can be part of goal reasoning frameworks (e.g., (Roberts et al.)). The paper showed how this strategy can be implemented by exploiting the same CSP encoding of the reconfiguration, and how this should be managed in order to infer minimal correcting sets (Marques-Silva et al. 2013).

The approach has been tested on a challenging domain such as a space exploration domain, but its applicability is not restricted to this domain. Many other robotic tasks could benefit of the proposed approach, since in many of them the need of adapting the plan execution to the resources constrains is very relevant.

In this perspective, from an experimental point of view, we would like to deepen the still preliminary experimental analysis on the goal reasoning facility, and to verify the applicability of this strategy (and also of ReCon) on a larger set of domains.

From a methodological point of view, the approach can be improved in a number of ways. A first important enhancement is the search for an optimal solution. In the current version, in fact, ReCon just finds one possible configuration that satisfies the global constraints. In general, one could be

interested in finding the configuration that optimizes a given objective function. Reasonably, the objective function could take into account the number of changes to action modalities; for instance, in some cases it is desirable to change the configuration as little as possible to improve the stability of the plan. Of course, the search for an optimal configuration is justified when the global constraints are not strict, and there is enough cpu time at disposal. Otherwise, sub-optimal strategies should be investigated.

References

Barták, R., and Toropila, D. 2010. Solving sequential planning problems via constraint satisfaction. *Fundam. Inf.* 99(2):125–145.

Barták, R.; Ĉepek, O.; and Hejna, M. 2008. Temporal reasoning in nested temporal networks with alternatives. In Fages, F.; Rossi, F.; and Soliman, S., eds., *Recent Advances in Constraints*, volume 5129 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 17–31.

Block, S. A.; Wehowsky, A. F.; and Williams, B. C. 2006. Robust execution of contingent, temporally flexible plans. In *Proc. of National Conference on Artificial Intelligence (AAAI-06): 802-808*.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems* 19(3):297–331.

Calisi, D.; Iocchi, L.; Nardi, D.; Scalzo, C.; and Ziparo, V. A. 2008. Context-based design of robotic systems. *Robotics and Autonomous Systems (RAS)* 56(11):992–1003.

Cesta, A., and Fratini, S. 2009. The timeline representation framework as a planning and scheduling software development environment. In *Proc. of P&S Special Interest Group Workshop (PLANSIG-10)*.

Conrad, P. R., and Williams, B. C. 2011. Drake: An efficient executive for temporal plans with choice. *Journal of Artificial Intelligence Research (JAIR)* 42:607–659.

Felfernig, A.; Schubert, M.; and Reiterer, S. 2013. Personalized diagnosis for over-constrained problems. In *Proc. of IJCAI-13*.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences* 18(2):231–271.

Garrido, A.; C., G.; and Onaindia, E. 2010. Anytime plan-adaptation for continuous planning. In *Proc. of P&S Special Interest Group Workshop (PLANSIG-10)*.

Gerevini, A., and Serina, I. 2010. Efficient plan adaptation through replanning windows and heuristic goals. *Fundamenta Informaticae* 102(3-4):287–323.

Gerevini, A.; Saetti, I.; and Serina, A. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence* 172(8-9):899–944.

Gerevini, A.; Saetti, A.; and Serina, I. 2012. Case-based planning for problems with real-valued fluents: Kernel functions for effective plan retrieval. In *Proc. of European Conference on AI (ECAI-12)*, 348–353.

Ghallab, M.; Nau, D.; and Traverso, P. 2014. The actor's view of automated planning and acting: A position paper. *Artificial Intelligence* 208(0):1 – 17.

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)* 20:291–341.

Liffiton, M. H., and Sakallah, K. A. 2008. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning* 40(1):1–33.

Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a csp. In *Proc. of International Conference on Artificial Intelligence (IJCAI-03)*, 954–960.

Marques-Silva, J.; Heras, F.; Janota, M.; Previti, A.; and Belov, A. 2013. On computing minimal correction subsets. In *IJCAI*.

Micalizio, R.; Scala, E.; and Torasso, P. 2011. Intelligent supervision for robust plan execution. In *LNCS 6954 of Associazione Italiana per Intelligenza Artificiale (AIxIA-11)*, 151–163.

Micalizio, R. 2013. Action failure recovery via model-based diagnosis and conformant planning. *Computational Intelligence* 29(2):233–280.

Muscettola, N. 1993. Hsts: Integrating planning and scheduling. Technical Report CMU-RI-TR-93-05, Robotics Institute, Pittsburgh, PA.

Narendra, J.; Rochart, G.; and Lorca, X. 2008. Choco: an open source java constraint programming library. In *Proc. of CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, 1–10.

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. 2009. Solve-and-robustify. *Journal of Scheduling* 12:299–314. 10.1007/s10951-008-0091-7.

Roberts, M.; Vattam, S.; Aha, D.; Apker, T.; Auslander, B.; and Wilson, M. Iterative goal refinement for robotics. In *Proc. of the Planning and Robotics Workshop (PLANROB-14) at ICAPS-14*.

Scala, E.; Micalizio, R.; and Torasso, P. 2014. Robust execution of rover plans via action modalities reconfiguration. In *Proc. of ICAART-14*, 142–152.

Scala, E. 2013a. Numeric kernel for reasoning about plans involving numeric fluents. In Baldoni, M.; Baroglio, C.; Boella, G.; and Micalizio, R., eds., *AI*IA 2013: Advances in Artificial Intelligence*, volume 8249 of *Lecture Notes in Computer Science*. 263–275.

Scala, E. 2013b. *Reconfiguration and Replanning for robust Execution of Plans Involving Continous and Consumable Resources*. Ph.D. Dissertation, Department of Computer Science - Turin.

van der Krogt, R., and de Weerdt, M. 2005. Plan repair as an extension of planning. In *Proc. International Conference on Automated Planning and Scheduling (ICAPS-05)*, 161–170.

Multi-Robot Planning and Execution for an Exploration Mission: a Case Study

Guillaume Infantes, Charles Lesire, Cédric Pralet
ONERA - The French Aerospace Lab, F-31055, Toulouse, France
{Guillaume.Infantes,Charles.Lesire,Cedric.Pralet}@onera.fr

Abstract

This paper presents the first steps of the treatment of a real-world robotic scenario consisting in exploring a large area using several heterogeneous autonomous robots. Addressing this scenario requires combining several components at the planning and execution levels. First, the scenario needs to be well modeled in order for a planning algorithm to come up with a realistic solution. This implies modeling temporal and spatial synchronization of activities between robots, as well as computing the duration of move activities using a precise terrain model. Then, in order to obtain a robust multi-agent executive layer, we need a robust hierarchical plan scheme that helps identifying appropriate plan repairs when, despite the quality of the various models, failures or delays occur. Finally, we need various algorithmic tools in order to obtain flexible plans of good quality.

Motivation

Automatic exploration of large and hazardous areas benefits from the use of multiple robots. Indeed, the use of several robots working in parallel allows the duration of missions to be drastically decreased, which may be useful to deal with crisis situations or with search and rescue missions, in which the response time is a key criterion. The deployed team of robots may be heterogeneous, to take advantage of the skills of different kinds of robots: flying robots can see over buildings and quickly cover large distances; ground robots can accurately map the environment; some robots may be able to enter buildings, overcome obstacles, and even cross flooded areas. Operators often use tools for defining the mission of robots offline, taking into account various operational constraints. Robots must then act autonomously at execution time in order to adapt their behavior to complex, dynamic and uncertain environments, and to perform replanning tasks if needed. Autonomy is especially useful when communications are intermittent or unreliable. In such cases, it is indeed impossible to permanently control each robot from a remote mission center. However, operators often need some feedback at regular time intervals, in order to know how the mission is progressing and to get data collected by robots.

Operational Scenario We consider the problem of exploring an area using heterogeneous autonomous robots, subject to the supervision of human operators . When defining the mission, human operators first define some zones to be observed by the robots. These zones are considered as *observation tasks* that will be allocated to the robots depending on their capabilities (some robots cannot see under trees, or cannot enter buildings). In this work, robots are assumed to be individually able to localize themselves, to plan trajectories, and to perform navigation tasks in the environment, the latter being possibly mapped online using robot sensors.

In the mission considered, human operators need to regularly monitor mission execution. Due to intermittent or unreliable communications between robots, and between robots and the operators, this online monitoring is defined as a time rate at which each robot has to report an execution status for its plan. This execution status may for instance contain the current robot position and the list of zones it has observed. Due to communication and motion constraints, aerial robots, which can move faster, are used to collect other robots data and to communicate these data to the operators. Reporting tasks correspond to temporal and spatial rendezvous, during which robots share information.

Approach Followed The main contribution of this paper lies in the integration of several components for tackling the operational scenario described above. The integrated components are:

- *Multi-agent, temporal and hierarchical planning*: we define a planning algorithm that computes plans of actions for the whole team of robots. Built plans are hierarchical, in the spirit of Hierarchical Task Networks (HTN (Erol, Hendler, and Nau 1994)). Elementary actions in these plans are move, observation and communication actions. These plans enforce temporal constraints coming from the environment model (time needed to move from points to points, or to observe zones) and from the mission requirements (periodic reporting to the human operator). To deal with communications, we propose an offline planning algorithm that includes communication tasks within the initial plans. We use an offline planning approach as a first step to coordinate vehicles which may not be permanently visible from the operation center. The algorithm

used is linked with external libraries that compute for instance durations of moves performed during the exploration, based on a terrain model.

- *Mission execution and repair*: based on embedded reactive architectures, plans are executed on each robot in a distributed way, each robot being in charge of its own tasks. During execution, disturbances may occur, requiring either an adaptation of plans (for instance when a robot is going to be late at a rendezvous), or more global repairs when the plan is no longer feasible. As communication may be unavailable for a global replanning, plans can be locally repaired through communication between a few close teammates (Gateau, Lesire, and Barbier 2013). In this paper, we only consider hard-coded repair rules used in case of failures. More advanced techniques using a deliberative architecture on-board each robot could be considered.

The paper is organized as follows. The next section presents a modeling of the scenario, and the way it is translated into constraints. The general scheme of the execution process (including plan adaptation and repair) is then presented. Afterwards, we present the offline planning algorithm, along with the constraint-based framework the algorithm is based upon. Finally, we present an evaluation of the algorithm on some basic examples, and we show first results on a real-world scenario involving three robots.

This work has been partially supported by the DGA funded Action project (http://action.onera.fr) and the ANCHORS project (http://www.anchors-project. eu), funded by the German Federal Ministry of Education and Research (BMBF) and the French National Research Agency (ANR).

Modeling the Scenario

Static Data

The mission consists in observing a set of n_z zones using n_r robots.

- The team of robots is $R = \{r_i\}_{1 \leq i \leq n_r}$.
- The set of zones is $Z = \{z_i\}_{1 \leq i \leq n_z}$.
- Each robot r_i can reach a set of navigation points $P(r_i)$.
- For each robot r_i, function O_i gives the navigation points from which a zone z_j can be observed by r_i. We have $O_i(z_j) \subseteq P(r_i)$.
- For communications, we consider a function C that gives, for each couple of robots (r_i, r_j) the set of points from which robots r_i and r_j can communicate. We have: $C(r_i, r_j) \subseteq P(r_i) \times P(r_j)$.

Dynamic State Description

At a given time, a robot r_i can be at one of its navigation points $p_j \in P(r_i)$, which is denoted by $at(r_i, p_j)$. A robot r_i can also be navigating between two locations. In normal execution, no action is possible until the robot reaches its destination.

Furthermore, a zone must be observed only once, so we define function $toObs : Z \rightarrow \{\top, \bot\}$ that expresses if a zone z_j is to be observed ($toObs(z_j) = \top$) or if z_j has already been observed ($toObs(z_j) = \bot$).

Task Model

Each robot $r_i \in R$ can perform three elementary tasks:

- $goto(r_i, p_j, p_k)$: navigation from point p_j to point p_k; the preconditions at the beginning of the task are $p_j, p_k \in P(r_i)$, and $at(r_i, p_j)$; as for effects, $at(r_i, p_k)$ becomes true at the end of the task, whereas $at(r_i, p_j)$ becomes false at the beginning of it;
- $obs(r_i, p_j, z_k)$: observation of zone z_k from point p_j; preconditions $p_j \in O_i(z_k)$ and $toObs(z_k) = \top$ must hold at the beginning of the task; condition $at(r_i, p_j)$ must also hold at the beginning and during the task; as an immediate effect, we have $toObs(z_k) = \bot$;
- $com(r_i, r_j, p_k, p_l)$: communication between r_i and r_j, located at positions p_k and p_l respectively; preconditions are $p_k \in P(r_i)$, $p_l \in P(r_j)$, and $(p_k, p_l) \in C(r_i, r_j)$; also, $at(r_i, p_k)$ and $at(r_j, p_l)$ must hold at the beginning and during the task; we do not consider explicit effects, as they will be dealt with in a dedicated approach.

We associate with every task t a starting time point $\delta_s(t)$ and a duration $dur(t)$. We then define the end time of task t as $\delta_e(t) = \delta_s(t) + dur(t)$.

Constraints

Along with this formulation, we consider more elaborate constraints over the actions. In order to obtain realistic plans, we need to integrate some real-world knowledge based on models of the environment, and to be able to precisely synchronize communications between robots.

Visibility Constraints An external library provides us with the $O_i(z_j)$ and $C(r_i, r_j)$ static data. More precisely, visibilities for observation and communication are obtained from terrain models and ray-tracing algorithms, taking into account (optical) sensor ranges and occlusions.

Temporal Constraints on Motions One key point for tackling a realistic scenario is to model the duration of moves. We thus use an external function $dur(goto(r_i, p_j, p_k))$ which provides us which an estimation of the duration of motions. This function can be implemented using any path planning algorithms (LaValle 2006) based on precise terrain models. For our experiments, the external library we use implements an A^* algorithm on discrete terrain models taking into account robots' motion capabilities.

Temporal Constraints on Other Actions We consider a duration for observation and communication tasks. For now, these durations are constant, but this can be easily lifted up.

Constraints on Communications For an operational scenario, one important need for the operator is to regularly monitor the states of the robots and the progress of the mission. In our case, as communications are not always possible, we need to enforce periodical communications, in order to detect within a given time if a robot has a problem, for

instance if it is lost or blocked by an obstacle and unable to reach its objective.

In order to do so, we add a set of constraints on communications, using a centralized communication scheme:

- the operator has to be given a complete update periodically (every Δ minutes);
- communications with the operator are very limited;
- one of the robot is preferred for centralizing communications with other robots and the operator, because it has more motion capabilities than the others; for example an Autonomous Aerial Vehicle (AAV).

Dividing the Plan into Chunks

We solve the mission planning problem by dividing the plan into chunks. Each chunk corresponds to a sequence of observations followed by a communication between all teammates and the operator. Splitting plans into such chunks allows the operators to regularly receive a complete feedback concerning the state of each robot, which can make mission monitoring easier.

For making mission monitoring easier again, we build hierarchical plans, containing actions with different levels of abstractions. Operators may go deeper in the hierarchy of actions in order to understand what the robots are doing. We first define some abstract tasks.

Definition 1. *($goto\_and\_obs$) A $goto\_and\_obs$ abstract task is made of a movement task g and an observation task o:*

- $g = goto(r, p_i, p_j)$ *and* $o = obs(r, p_j, z_k)$*;*
- $r \in R$ *is the robot performing the tasks;*
- $p_j \in P(r)$ *is the point from where* $z_k \in Z$ *is observed;*
- $\delta_e(g) = \delta_s(o)$.

This abstract task is denoted as $goto\_and\_obs(r, p_i, p_j, z_k)$.

Definition 2. *($goto\_and\_com$) A $goto\_and\_com$ abstract task is made of two tasks g and c such that:*

- $g = goto(r_i, p_u, p_v)$ *and* $c = com(r_i, r_j, p_v, p_w)$*;*
- $r_i \in R$ *is the robot performing the tasks;*
- $p_v \in P(r_i)$ *is the point from where a communication is possible to* $p_w \in P(r_j)$*;*
- $\delta_e(g) = \delta_s(c)$.

This abstract task is denoted as $goto\_and\_com(r_i, r_j, p_u, p_v, p_w)$.

Now we can define a chunk as a sequence of $goto\_and\_obs$ for each robot, followed by synchronized $goto\_and\_com$ for all robots.

Definition 3 (Chunk)**.** *Let r_m denote the robot in charge of the centralization of the communications (m for "master"), and let op denote the operator. A chunk c is composed of:*

- *for each $r_i \in R$, a sequence of abstract tasks $goto\_and\_obs(r_i, p_j, p_{j+1}, z_k)$;*
- *for each $r_i \in R$, $i \neq m$, a task $goto\_and\_com(r_i, r_m, p_u, p_v, p_w)$ for robot r_i and a symmetric task $goto\_and\_com(r_m, r_i, p_x, p_w, p_v)$ for robot r_m; communications are synchronized, i.e. $\delta_s(com(r_i, r_m, p_v, p_w)) = \delta_s(com(r_m, r_i, p_w, p_v))$*
- *a task $goto\_and\_com(r_m, op, p_y, p_z, p_{op})$ at the end of the chunk, with p_{op} the point where the operator is.*

We define $dur(c)$ as the temporal distance between the beginning of the first $goto\_and\_obs$ task of the chunk and the end of the communication with the operator. The chunk is said to be valid if and only if $dur(c) \leq \Delta$, where Δ is the communication period set by the human operator.

The intuition is that a chunk is a refinement method in the HTN framework, augmented with temporal constraints. If communication period Δ is too small, then it will be impossible to include some observations into plans. A schematic graphical view of a chunk is given in Figure 1.

Definition 4 (Chunked Plan)**.** *A chunked plan $\mathcal{C}$ is a sequence of chunks $(c_i)_{i \geq 0}$ such that at the end of the execution we have $\forall z_k \in Z : toObs(z_k) = \bot$.*

The planning algorithm must ensure that given the chosen observations, the duration of any chunk does not exceed the required communication period (Δ). It must also minimize the number of chunks and, as a side effect, the total duration of the mission.

Supervising Execution

Plan Representation

Hierarchical and Temporal Tree of Tasks (HT^3) We represent the plan as a hierarchical tree of tasks along with their temporal execution windows. As said before, representing plans in such a way can make plans more readable for the operators. We follow the common hierarchical model of HTNs (Erol, Hendler, and Nau 1994) to model the plan, and we include some temporal information (Bresina et al. 2005) as well as allocation of tasks to robots.

Definition 5 (HT^3)**.** *An HT^3 plan $\mathcal{P} = (R, T, M, \mathcal{A}, \mathcal{I}, t_r)$ is defined by:*

- *a set of robots $R = \{r_i, 1 \leq i \leq n_r\}$;*
- *a set of tasks $T = T_e \cup_{\neq} T_a$, where T_e is a set of elementary tasks representing robots' actions, and T_a is a set of abstract tasks;*
- *a decomposition function (or set of methods) $M : T_a \rightarrow 2^{(T_a \cup T_e)} \times \{\rightarrow, \sim\}$ that associates with each abstract task a set of tasks st and a relation rel between the elements of st; rel is either a non-ordered relation ($\sim$, tasks of st can be executed in any order) or a sequence ($\rightarrow$, tasks of st must be executed in a specific order);*
- *an allocation function $\mathcal{A} : T \rightarrow 2^R$ that associates with each task a set of robots; for elementary tasks, only one robot is performing the task : $\forall t \in T_e, \#\mathcal{A}(t) = 1$;*
- *an interval function $\mathcal{I}$ that associates with each task t a time interval $\mathcal{I}(t) = [\delta^-, \delta^+]$ such that δ^- and δ^+ are the earliest and latest times to start the execution of the task;*
- *a root task $t_r \in T$.*

A plan $\mathcal{P}$ is valid only if it has no cycles (e.g., abstract tasks being child of themselves). It can then be represented as a tree, alternating tasks and methods. Note that contrarily to HTNs, HT^3 do not allow any choice in the decomposition of abstract tasks, which makes them directly executable.

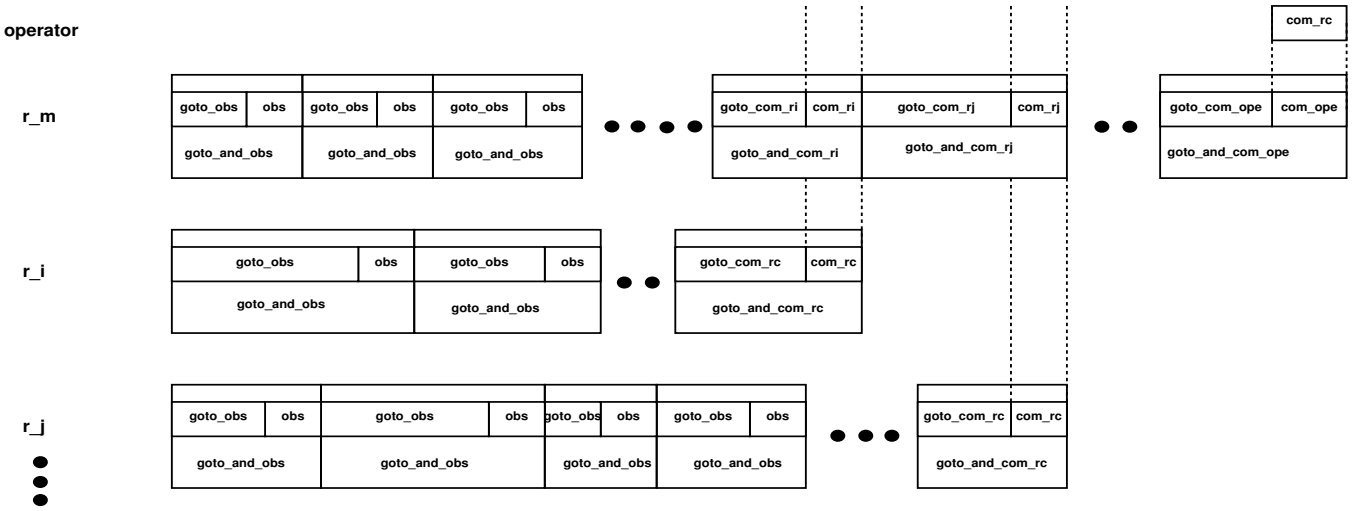


Figure 1: General scheme of a chunk

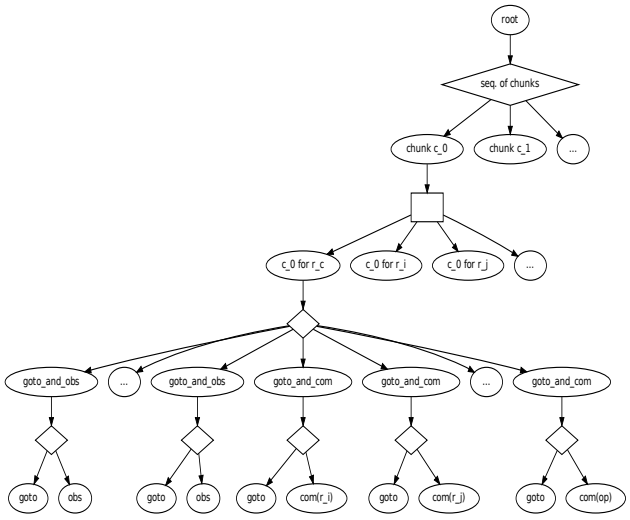


Figure 2: HT^3 built from a sequence of chunks. Ovals, diamonds, rectangles respectively represent tasks, sequential methods, unordered methods. For clarity, allocation and interval functions are not represented.

Chunked Plans as HT^3 Building an HT^3 plan (Def. 5) from a chunked plan (Def. 4) is quite straightforward: we first define a root task t_r, containing a sequential method with the sequence of chunks as children. Then each chunk is decomposed into a non-ordered method containing the tasks of each robot. Finally, the chunk for each robot is sequentially decomposed into abstract tasks $goto\_and\_obs$ and $goto\_and\_com$, themselves respectively decomposed into $goto$ and obs, and $goto$ and com elementary actions. Figure 2 shows the generic hierarchical pattern of a plan.

Algorithm 1 Execution of hierarchical and temporal plans

Require: $\mathcal{P}$ the global team plan, $r \in R$ the robot that locally executes the plan

```
1: procedure SCANTASK(t, r, P)
2:     CHECKTIME(I(t))
3:     if t ∈ T_e then EXECUTE(t)
4:     else                                   ▷ t ∈ T_a
5:         st, rel ← M(t)
6:         Sort st according to rel
7:         for all t' ∈ st do
8:             if r ∈ A(t') then SCANTASK(t', r, P)
```

Plan Execution

Each robot executes plan $\mathcal{P}$ following the procedure detailed in Algorithm 1. The plan is executed by each robot in a depth first (i.e. descending from abstract tasks to elementary tasks) and left first (i.e. enforcing the ordered relations between tasks) manner. Procedure SCANTASK is initially called with t_r (the root task) as a parameter.

Procedure CHECKTIME used at line 2 checks that the current execution time is in the allowed window of start times for task t. If the task is early, we wait for the earliest starting time of the task. If the task is on-time, we start it immediately. If the task is late, a repair is needed (see next paragraph).

Procedure EXECUTE used at line 3 triggers the execution of an elementary task on the robot. In practice, this is achieved by calling a service on the robot control architecture, in order to move the robot, make observations, or communicate with other robots.

Plan Repair

At execution, two kinds of disturbances are considered: lateness of tasks and execution failures of elementary actions.

Lateness During Execution In the CHECKTIME procedure (line 2), the task to execute may be detected as late, i.e. the latest time at which it should have started has passed. Then, the execution is inconsistent regarding the plan.

As communication is unreliable and not persistent, we cannot count on a global repair. Therefore, the repair process is driven by the key feature which ensures the success of the mission: the communication constraint between robots. We therefore try to secure communication tasks. For this purpose, the repair process is a simple hard-coded rule that removes $goto\_and\_obs$ tasks from the plan of the late robot until the current chunk becomes temporally consistent again. The new plan is executed until the end of the chunk, where a communication between all robots is possible (by using master robot r_m). At the end of the chunk, the operator knows the set of observations removed during repairs. He/she can then reallocate these observations and transmit the new plans to all robots through master robot r_m.

Failures of Execution Execution disturbances may also come from issues in the execution of tasks by the control architecture of the robot. For instance a mobile ground robot may face some obstacles during its movements. If it reaches the aimed point, the next task may be late due to the time needed to overcome the obstacle (previous lateness case). But the robot may even not reach the point, for instance because no path exists (at least in the robot map) to join this point. In these situations, which are considered as failures, we use some precomputed parametered local hierarchical plans that are adapted for repairing a set of predefined failures. When a failure occurs, depending on the task that has failed and of the cause of the failure, we replace the failing task with the appropriate local plan. More advanced plan repair techniques could be considered. In our experiments, three kinds of parametered local plans are considered:

- parametered backup trajectories to find communication opportunities with the master robot (to ask for help);
- map sharing between robots (to update the map of the lost robot);
- relative localization between robots (to update the position of the lost robot).

As these parametered local plans are represented as "local" HT[3], they can be directly inserted within the current plan and executed without any hack in the execution procedure.

Note that this paper does not provide repair rules for all possible lateness issues and all possible failures. It just describes a first step in the definition of hard-coded repair rules for the operational scenario we consider. Additional work on this point is left for future work. In particular, rules should be defined for tackling the case where master robot r_m itself fails, or cases in which removing observation tasks does not suffice to satisfy the chunk duration constraint again.

Planning

In this section, we describe the planning model implementation and the planning algorithm used to build mission plans offline.

Using a Generic Framework: InCELL

In order to state all temporal constraints of the problem and to handle them efficiently, we rely on the InCELL framework (Pralet and Verfaillie 2013). InCELL is inspired by *Constraint-based Local Search* (CLS (Hentenryck and Michel 2005)). In CLS, the user defines a model of its problem in terms of decision variables, constraints, and optimization criterion. For the multi-robot exploration mission, (1) *decision variables* correspond to the sequence of tasks to be performed by each robot, (2) *constraints* are either temporal constraints (activity durations and communication deadlines) or spatial constraints (zone observation and robot communication from possible locations), and (3) the *criterion* is to minimize the duration of the mission.

In CLS, the user also defines a local search procedure over complete variable assignments (where every decision variable is assigned). Such a local search procedure corresponds to a sequence of local moves. For the multi-robot exploration problem, examples of local moves are the addition/removal of a chunk and the addition/removal of an observation task for a robot inside a chunk.

Because the speed of local moves is one of the keys to local search success, CLS uses so-called *invariants*, which allow expressions and constraints to be quickly evaluated after each move. Invariants correspond to one-way constraints $[x_1, \ldots, x_n] \leftarrow f(y_1, \ldots, y_p)$, where $x_1, \ldots, x_n$ (the outputs) are variables whose assignment is a function of other variables $y_1, \ldots, y_p$ (the inputs). Invariant outputs are incrementally (quickly) reevaluated upon small changes in the inputs. In particular, InCELL models temporal constraints as invariants and uses incremental *Simple Temporal Network* (STN (Dechter, Meiri, and Pearl 1991)) techniques to efficiently maintain earliest/latest consistent times for activities.

Task Modeling InCELL allows tasks to be modeled based on the notion of *interval*. An interval is defined by two timepoints representing the start and the end of the task, and by one boolean variable representing the presence of the task in the plan. Elementary tasks of the multi-robot exploration problem (moves, observations, communications) are represented as InCELL intervals.

For the multi-robot exploration mission, goal activities are observations and communications, whereas setup activities are motions that occur between goal activities. Tasks goto_and_obs and goto_and_com represented in Figure 1 are modeled as intervals composed respectively of two subintervals [*goto_obs , obs*] and [*goto_com, com*]. Doing so, the hierarchical structure of plans is explicitly taken into account in the model.

The duration of each goal activity is constant in our case. The duration of a move towards a goal activity *Act* depends on the goal activity *Act* itself and on the goal activity *preAct* preceding the move. If the current and previous activities *Act* and *preAct* are provided, the InCELL invariant evaluator automatically calls the external terrain-aware function that computes estimations of move durations.

Chunk Modeling Beyond intervals, the multi-robot exploration model also uses the notion of sequence of contiguous intervals available in InCELL. Using this modeling

feature, it is easy to implement chunks as $n_r + 1$ sequences of contiguous intervals, one for each robot plus one for the operator. When inserting a new task in the middle of a sequence, one only needs to update the "previous activity" feature for the inserted task and for the task following it. The InCELL invariant in charge of managing temporal aspects then automatically updates earliest/latest times of all time-points of the problem, using incremental STN techniques. The chunk itself is a specialization of a temporal interval, so it also has start and end time-points.

Mission Constraints as InCELL Invariants The InCELL invariant managing temporal aspects takes as input all temporal constraints of the problem inside a chunk:

- equality of communication starts and communication ends for the two robots involved in a communication task;
- start time of the chunk equal to the start time of the first move of the chunk;
- end time of the communication with the operator equal to the end time of the chunk;
- bounded temporal distance between the start and the end of a chunk, in order to enforce communication period Δ.

Algorithm

On top of the InCELL model, we define an algorithm that allocates observation tasks within chunks. This algorithm combines: (1) a constructive greedy search phase, which produces a first exploration and communication plan; (2) a local search phase that improves on the plan found at the first phase.

Greedy Search We first use a greedy search that tries to put as many as possible observations inside a chunk, and when constraints are violated, a new chunk is created, and the process iterates until all zones are scheduled for observation. The pseudo-code is detailed in Algorithm 2.

In this algorithm, the main loop iterates until no more zones are to be observed. Inside this main loop, a chunk is first allocated at line 8, already containing $goto\_and\_com$ tasks, as described in the modeling section. Then a robot is selected at line 9. Several strategies are possible, we implemented a random choice giving more weight to slower robots, in order for them to be able to choose first their objectives, because in our scenarios all robots start from a close location, and we do not want the slower ones to go very far to achieve their first objectives. This procedure return $\emptyset$ when all robots are marked as full, that is when no robot can accept a new observation in the current chunk.

We then enter a second loop for filling current chunk c_i. First, the closest observation is selected for robot r_j at line 11. This is done on the basis of the navigation points $P(r_j)$ from which a zone that has not yet been observed can be observed, and of a heuristic implemented as an InCELL invariant (see below). We implemented both a simple Euclidean distance and the real distance as given by the external duration function used by the constraint checker.

The selected observation is inserted at the end of the chunk, just before the communication tasks (line 12); then,

Algorithm 2 Greedy search for allocating observation tasks

Require:
1: Z the set of zones to observe
2: R the set of robots
3: P the function giving navigation points
4: O the function giving points for observing zones
5: C the function giving pair of points for communication

6: **procedure** GREEDYSEARCH(Z,R,P,O,C)
7: **while** $Z \neq \emptyset$ **do**
8: c_i = ALLOCATENEWCHUNK
9: $r_j \leftarrow$ SELECTROBOT(R)
10: **while** $r_j \neq \emptyset$ **do**
11: $(z_k, p_l) \leftarrow$ SELECTOBS($Z, P(r_j), O, p_{l-1}$)
12: INSERT($c_i, (z_k, p_l), r_j$)
13: UPDATECOMLOC(c_i, C, P)
14: ok = EVALUATECONSTRAINTS
15: **if** $\neg$ ok **then**
16: CHANGECOMORDER(c_i, C, P)
17: UPDATECOMLOC(c_i, C, P)
18: ok = EVALUATECONSTRAINTS
19: **if** $\neg$ ok **then**
20: REVERTCHANGES
21: TAGASFULL(r_j, c_i)
22: **if** ok **then**
23: $Z \leftarrow Z \setminus \{z_k\}$
24: $r_j \leftarrow$ SELECTROBOT(R)

the communication locations are updated, by choosing for the slower robot the communication point that is the closest from the newly inserted observation task (line 13). A location for the other robot involved in the communication is chosen among the restricted possible ones, also based on its last objective so far. Then the InCELL model is evaluated at line 14.

If the insertion fails (lines 16-17), we try to swap some communication activities for master robot r_m (these activities are initially randomly ordered). We also try to change communication locations. If insertion is still impossible, we discard changes and mark the selected robot as full for the current chunk, meaning that it does not accept new observations in the current chunk.

If insertion is possible, we mark the zone as observed (line 23), we select a robot and we iterate the inner loop again. This loop ends as soon as all robots are full for the current chunk.

Heuristics as Invariants InCELL offers various high level invariants, including the *argmin* invariant used for selecting items into a set based on some criterion. We thus use a selection based on the argmin invariant to maintain the closest observation candidates for any robot, as well as the closest communication location candidates. These invariants allow a transparent heuristic computation, used in the SELECTOBS and UPDATECOMLOC procedures (lines 11 and 13).

Local Search While greedy search aims at minimizing the number of chunks (and thus the total number of mandatory communications), it has a major drawback: the last chunk is generally very inefficient, because of the priority given to the slower robots. It may involve only a few observations by the slower robot, while other faster robots do not have any. To overcome this issue, we perform a local search from the solution produced by the greedy search.

The lower levels local search operators are to *remove* an observation from any chunk (including the corresponding goto), and to *insert* a zone to observe anywhere in the plan, trying every observation location for a given zone.

Over these two basic local moves, we implemented:

- a *bestInsert* procedure, which tries in turn every possible insertion for a given zone to be observed (in every chunk), and returns an updated plan with the best insertion, in terms of earliest end time of the global mission;
- a *moveInSequence* procedure, that removes and tries to *bestInsert* every observation in turn, this for a given robot.

We also implemented higher level local moves, namely:

- *2-opt* (Croes 1958), which tries permutations of pairs of observations and inverts orders in-between;
- *relocate* (Salvelsbergh 1992), which first removes an observation from a robot, second tries to *bestInsert* it in the plan of another robot, third applies *2-opt*, and fourth applies *moveInSequence*; it does so for every observation, as long as there are improvements in the global earliest end time of the mission.

Other moves based on the computation of critical paths could also be used to compact the obtained schedules.

Evaluating Planning and Local Search

We first show some planning results on simple scenarios, to give a first idea of the efficiency of the approach.

First Example We first consider a 100 m $\times$ 100 m area containing 25 zones to be observed, and a temporal constraint on communications allowing the whole mission to be executed in only one chunk (namely 5 minutes). Figure 3a shows the trajectories for 1 Autonomous Aerial Vehicle (AAV) and 2 Autonomous Ground Vehicles (AGVs) after greedy search, without any local optimization. The earliest end time of the mission is 299.2 seconds. After local search, we non surprisingly obtain better trajectories, shown on Figure 3b, and the earliest end time of the mission lowers to 214.7 seconds. In the first case, the greedy search gives high priority to the two AGVs, so that they have respectively 11 and 12 objectives to observe, while the AAV, which is much faster, only has two. After local search, the AAV has 12 objectives, and the AGVs 7 and 6, respectively, leading to a much better distribution of tasks over time.

For this simple problem, on an Intel Core 2 Duo 3GHz-4GBRAM, the construction and initialization of the InCELL model took 780ms, and greedy search time is 161 ms. The local search took 2833 ms.

Other Examples A second example considers the very same problem but with a 3-minute constraint for chunk durations, so that the mission cannot be achieved in only one chunk (it takes 2). Before local search, the earliest end time is 358.6 seconds. After local search, it lowers to 291.2 seconds. In this case, the local search time raises to 9141 ms.

To give an idea for larger scenarios, if we consider 100 zones to observe, the greedy search takes 682 ms, while the local search raises to approximately two minutes. We thus have a first solution very quickly, and the local search can be used as an anytime algorithm, giving better solution as time passes. Figure 3c shows the evolution of the earliest end time of the mission wrt. computation times for 100 zones to observe.

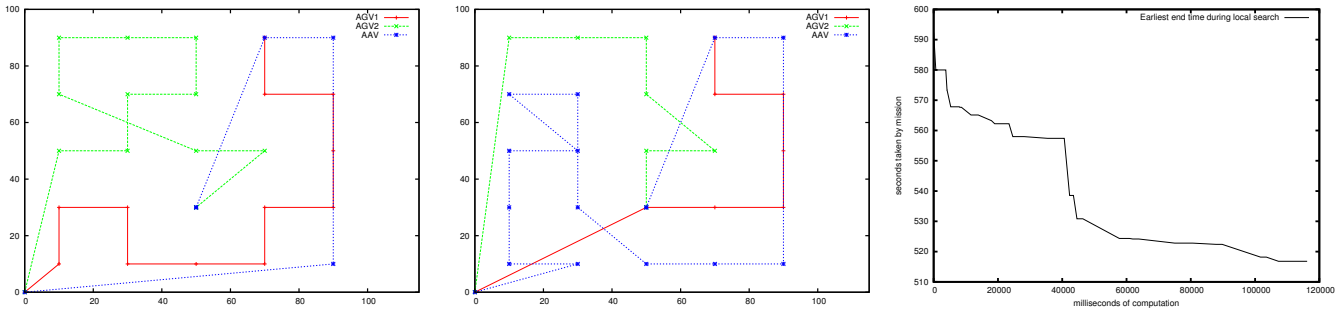
Complete Scenario

The complete scenario demonstrated fall 2013 involved one autonomous aerial vehicle and two autonomous ground vehicles. The area to explore was 28800 square meters wide, including 72 zones to observe. Figure 4a shows the area used for the experiments, for which a 3D model has been built in order to compute visibilities (for observations and communications) as well as durations of movements. The operator had to be given a complete update every 5 minutes (maximum size of a chunk). As flight authorizations are required to use our experimental platforms, only a few tests were performed in real conditions.

The InCELL model contains 31154 invariants. It is built and initialized in about 7 seconds. The resolution based on greedy and local search takes about 6 minutes, the best solution being obtained only after 3 minutes of computations. The huge difference with simple examples, that are about the same size, is mainly due to the calls to the precise computation of motion times, that are very slow. We implemented a cache in order to limit this impact.

Figure 4b shows the trajectories computed for this real-world mission. The duration of the mission plan obtained is 472 seconds. This duration is mainly due to the fact that the AAV has many more observations to do than the AGVs (the AGVs fill their objectives during the first chunk, while the AAV needs two). One can also notice that the observation tasks assigned to the two AGVs have a particular spatial repartition. This is due to the fact that their possible positions are very constrained: they must remain on the road since their navigation on the field can be problematic depending on the precise state of the terrain (height of the grass, wetness of the soil).

The execution procedure has been integrated on the three robots, as a separate program calling each robot's services to realize elementary actions. The aerial autonomous robot is a Yamaha Rmax helicopter that navigates based on a GPS sensor, and embeds a software architecture based on Orocos (Soetens and Bruyninckx 2005), in which actions are triggered by executing supervision state machines specified in the rFSM language (Klotzbücher and Bruyninckx 2012). The ground robots are Segway-based robots integrating lidar sensors for map building, cameras and inertial sensors used for localization, and embedding a Genom-based (Mallet,



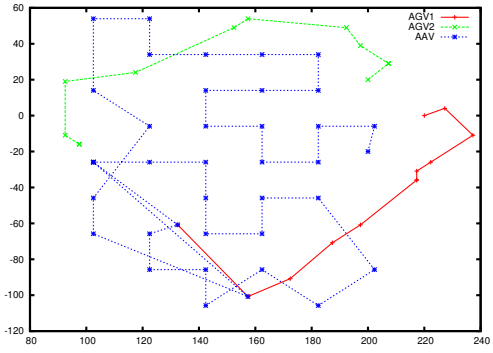
(a) Example 1 before local search (b) Example 1 after local search (c) Earliest end time (100 zones)

Figure 3: Plans produced on simple examples by the offline planning algorithm

Pasteur, and Herrb 2010) software architecture, in which actions are triggered by executing specific agents of the ROAR framework (Degroote and Lacroix 2011).

In fall 2013, we demonstrated the execution of the nominal plan, along with the management of some failures (one of the two ground robots get lost and, following hard-coded repair rules, asked for the Rmax helicopter to map the environment around the ground robot). We also demonstrated the replanning process including reallocation of the unexplored zones. We plan to do some complementary experiments in the future, in order to demonstrate the management of lateness of execution, by introducing disturbances at any moment in the mission execution.

(a) Map of the experiments area



(b) Trajectories after greedy and local search

Figure 4: Real-world scenario

Related Work

Some works focus on allocating exploration tasks to several robots, but either do not consider communication constraints (using frontier-like exploration (Hourani, Hauck, and Jeschke 2013) or a segmentation of the environment (Wurm, Stachniss, and Burgard 2008)), or try to maintain communication capabilities at any time by deploying a network infrastructure (Pei and Mutka 2012; Abichandani, Benson, and Kam 2013). Some approaches use opportunistic communications to optimize the plan (Sung, Ayanian, and Rus 2013), but do not enforce them. These approaches do not consider time constraints between tasks, and when synchronization is explicitly modeled, it is focused on spatial synchronization (Coltin and Veloso 2012). Other approaches propose mechanisms to maintain the plan consistency at execution. In (Kaminka et al. 2007), robots regularly communicate to update temporal constraints between their tasks in order to maintain the global plan consistency. In the exploration mission considered in (Korsah et al. 2012), offline task scheduling and online plan flexibility are combined, and robots adapt their plans by exchanging messages for satisfying task constraints again. However, associated communication tasks are not explicitly included within plans. Moreover, these tasks are considered as not subject to failures.

In another direction, multi-robot task scheduling deals with time constraints such as task precedence or synchronization (Zhang and Parker 2013). In (Ponda et al. 2010; Luo, Chakraborty, and Sycara 2013), tasks are scheduled using an auction algorithm to minimize their delays. Mixed Integer Linear Programming (MILP) is used in (Koes, Nourbakhsh, and Sycara 2006) to solve task allocation problems with constraints modeled using Allen's algebra, and in (Mathew, Smith, and Waslander 2013) to find trajectories of robots that must meet the already planned trajectories of robots to be recharged. In these works, once tasks are scheduled, no communication occurs to share information and maintain plan consistency.

In probabilistic domains, (Wu, Zilberstein, and Chen 2011) proposes to broadcast history of actions and observations when an inconsistency is detected between the current observation and the belief state of a local POMDP. (Matignon, Jeanpierre, and Mouaddib 2012) uses

a DecMDP model with communication uncertainty, while (Spaan and Melo 2008) defines local interactions within a so-called IDMG model. While these approaches generate policies which include communication tasks, they do not integrate temporal constraints between tasks.

Conclusion and Future Work

We presented in this paper the high-level aspects of a complete multi-robot exploration mission, from mission modeling to execution and repair, including planning algorithms taking into account various constraints. We rely on the robust generic framework InCELL for planning algorithms, as well as a HTN-like paradigm (augmented with temporal aspects) for solution scheme, distributed execution and repair.

This work will continue in several directions. First, at model level, we want to take into account resources like energy, but also to do time-dependent scheduling. A typical case is that for optical sensors, it might be useful to be able to select observations locations with respect to the precise time of the action, in order not to have sun on sight for instance. Tasks should also be possibly done in parallel. For instance an observation should not be simply to take a picture of a location from a given position, but grab a whole video flow for a given time, this while moving. Similarly, communication and observation could be performed in parallel. We also would like to use a good criterion for plans, including much more information than only the earliest global end time. This includes a trade-off with the flexibility, but also variable costs of motions, variable values of interest for zones. Finally, we would like to stop discretizing the mission over predefined observation zones.

At planning level, we are also working on a very different approach for planning with temporal HTNs, that should be more generic and allow a user (maybe directly the operator) to specify the chunk-like decomposition, instead of the current fixed one.

At execution and especially at repair level, we plan to make the planning model "alive" on the different robots and feed it with real execution times in order to detect temporal inconsistencies much earlier, and replan as soon as possible. While deferring observation tasks for a given robot is an easy solution, ensuring global consistency and quality of the plan is quite a challenge. As previously mentioned, we plan to extend the set of failure cases which can be handled by the repair process. Last, we would also like to integrate the hand-written repair methods into the main planning loop, instead of leaving them only at the supervision level.

References

Abichandani, P.; Benson, H.; and Kam, M. 2013. Robust Communication Connectivity for Multi-Robot Path Coordination using Mixed Integer Nonlinear Programming: Formulation and Feasibility Analysis. In *International Conference on Robotics and Automation (ICRA)*.

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity planning for the mars exploration rovers. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 40–49.

Coltin, B., and Veloso, M. 2012. Optimizing for Transfers in a Multi-Vehicle Collection and Delivery Problem. In *International Symposium on Distributed Autonomous Robotic Systems (DARS)*.

Croes, G. A. 1958. A method for solving traveling salesman problems. *Operations Research* 6:791–812.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

Degroote, A., and Lacroix, S. 2011. ROAR: Resource oriented agent architecture for the autonomy of robots. In *International Conference on Robotics and Automation (ICRA)*.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN Planning: Complexity and Expressivity. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Gateau, T.; Lesire, C.; and Barbier, M. 2013. HiDDeN: Cooperative Plan Execution and Repair for Heterogeneous Robots in Dynamic Environments. In *International Conference on Intelligent Robots and Systems (IROS)*.

Hentenryck, P. V., and Michel, L. 2005. *Constraint-based Local Search*. MIT Press.

Hourani, H.; Hauck, E.; and Jeschke, S. 2013. Serendipity Rendezvous as a Mitigation of Explorations Interruptibility for a Team of Robots. In *International Conference on Robotics and Automation (ICRA)*.

Kaminka, G.; Yakir, A.; Erusalimchik, D.; and Cohen-Nov, N. 2007. Towards collaborative task and team maintenance. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*.

Klotzbücher, M., and Bruyninckx, H. 2012. Coordinating Robotic Tasks and Systems with rFSM Statecharts. *Journal of Software Engineering for Robotics (JOSER)* 3(1):28–56.

Koes, M.; Nourbakhsh, I.; and Sycara, K. 2006. Constraint optimization coordination architecture for search and rescue robotics. In *International Conference on Robotics and Automation (ICRA)*.

Korsah, A.; Kannan, B.; Browning, B.; Stentz, A.; and Dias, B. 2012. xBots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies. In *International Conference on Robotics and Automation (ICRA)*.

LaValle, S. 2006. *Planning Algorithms*. Cambridge University Press.

Luo, L.; Chakraborty, N.; and Sycara, K. 2013. Distributed Algorithm Design for Multi-Robot Task Assignment with Deadlines for Tasks. In *International Conference on Robotics and Automation (ICRA)*.

Mallet, A.; Pasteur, C.; and Herrb, M. 2010. GenoM3: Building middleware-independent robotic components. In *International Conference on Robotics and Automation (ICRA)*.

Mathew, N.; Smith, S. L.; and Waslander, S. L. 2013. A Graph-Based Approach to Multi-Robot Rendezvous for Recharging in Persistent Tasks. In *International Conference on Robotics and Automation (ICRA)*.

Matignon, L.; Jeanpierre, L.; and Mouaddib, A.-I. 2012. Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Pei, Y., and Mutka, M. 2012. Steiner traveler: Relay deployment for remote sensing in heterogeneous multi-robot exploration. In *International Conference on Robotics and Automation (ICRA)*.

Ponda, S.; Redding, J.; Choi, H.-L.; How, J.; Vavrina, M.; and Vian, J. 2010. Decentralized Planning for Complex Missions with Dynamic Communication Constraints. In *American Control Conference (ACC)*.

Pralet, C., and Verfaillie, G. 2013. Dynamic online planning and scheduling using a static invariant-based evaluation model. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Salvelsbergh, M. W. P. 1992. The vehicle routing problem with time windows: Minimizing route duration. *Journal on Computing* 4:146–154.

Soetens, P., and Bruyninckx, H. 2005. Realtime hybrid task-based control for robots and machine tools. In *International Conference on Robotics and Automation (ICRA)*.

Spaan, M., and Melo, F. 2008. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*.

Sung, C.; Ayanian, N.; and Rus, D. 2013. Improving the Performance of Multi-Robot Systems by Task Switching. In *International Conference on Robotics and Automation (ICRA)*.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175:487–511.

Wurm, K.; Stachniss, C.; and Burgard, W. 2008. Coordinated multi-robot exploration using a segmentation of the environment. In *International Conference on Intelligent Robots and Systems (IROS)*.

Zhang, Y., and Parker, L. 2013. Multi-Robot Task Scheduling. In *International Conference on Robotics and Automation (ICRA)*.

Planning and Scheduling Single and Multi-Person Activities in Retirement Home Settings for a Group of Robots

Tiago Vaquero and **Goldie Nejat** and **J. Christopher Beck**
Department of Mechanical and Industrial Engineering, University of Toronto,
5 King's College Road, Toronto, Ontario, Canada M5S 3G8
{tvaquero, nejat, jcb}@mie.utoronto.ca

Abstract

Automated planning and scheduling (P&S) technology has been increasingly investigated and applied to various robotics applications. We introduce a challenging P&S problem in which multiple social robots must autonomously organize and facilitate human-robot interactions for one-on-one telepresence sessions and multi-user Bingo games. These activities need to take place throughout the day based on the individual availabilities of the residents living in a retirement home. We utilize a domain-independent P&S approach for this problem, studying different variations of a PDDL model and the performance of state-of-the-art temporal planners in five different scenarios. We demonstrate the modeling challenges and technological gap in domain-independent P&S technology for such real-world robot problems. In particular, modeling a combination of metric quantities, resources, temporal availability of residents and time constraints on cascading actions is non-trivial. Moreover, we show that the available temporal planners perform poorly on the problem and struggle with the optimization aspects of such real-world scenarios.

Introduction

Due to the rapid aging of the world's population and the shortage in healthcare professionals, robotic technologies are being increasingly developed to engage the elderly in cognitively and socially stimulating activities in eldercare environments (Pineau et al. 2003; Kidd, Taggart, and Turkle 2006; Fasola and Mataric 2012; McColl, Louie, and Nejat 2013). While some of this work has incorporated automated planning and scheduling (P&S) (Pineau et al. 2003; Pollack 2005; Cesta et al. 2011), the majority of the existing research in robotics and P&S in eldercare environments has focused on the human-robot interaction (HRI) activities of a single robot in one-on-one activities with a single user. Only a handful of works have considered robots interacting with multiple people at the same time, e.g., (Montemerlo et al. 2002). However, these robots have not actively distinguished between users to provide personalized interactions during multi-user activities. Given the variety of users' abilities and availabilities, multi-user assistance activities require robots to plan, schedule, and customize their HRI interactions to the needs, time constraints, availability and preferences of each individual during the day. An environment with multiple users, multiple robots, and single- and multi-person HRI activities has not been addressed in the P&S literature.

In this paper, we introduce such an environment and the associated P&S problem. A set of robots has to search for and interact with multiple residents living in a retirement home to perform a set of telepresence sessions (single-person activity), Bingo games (multi-person activity), and reminder deliveries. In addition, such activities deplete a robot's batteries and so a recharging activity may also be necessary.

The proposed problem provides a complex combination of reasoning about actions, resources (e.g., the robots), time windows (e.g., user availability), temporal constraints (e.g., activity deadlines), metric quantities (e.g., battery level), and optimization (e.g., maximizing the number of residents taking part in a Bingo game). Since the 1980s there has been a recurring discussion in the literature regarding the challenges of combining these elements, which have often been investigated independently (Fox 1999; Smith, Frank, and Jonsson 2000). However, developing solvers for P&S applications that include these features is still an open challenge.

The novelty of this work lies in: 1) presenting a new P&S problem for assistive robotics in retirement homes that considers multiple robots, users and user schedules, as well as single-user and multi-user HRI activities, and 2) an investigation of the state-of-the-art domain-independent temporal planners to solve the proposed problem.

Background

Our long-term project is the deployment of intelligent human-like mobile robots in retirement homes to engage residents daily in stimulating recreational activities (Louie, Han, and Nejat 2013; Louie et al. 2014). We use the robotic platform H20 from Dr Robot (Dr Robot 2014) and have designed the robot to: 1) navigate using a laser range finder and 3D depth sensors, 2) detect users with 2D cameras, and 3) interact with users through speech, gestures, and a touch screen. While the implementation of the robot behaviors addresses real robotics challenges (e.g., sensing, HRI, person recognition), herein we focus on the planning and scheduling of the daily activities of the social robots. Details of robot implementation can be found in (Louie et al. 2014).

We focus on two representative activities: *telepresence* and *Bingo*. In the former, the robot autonomously navigates to the user in his/her private room, prompts the user for the video call, starts the call and tracks the user during the session. For the Bingo game, the robot autonomously finds and reminds participants about the game prior to its start and then navigates to a specified location to conduct the game. During Bingo, the robot acts as the game facilitator, calling out numbers, verifying Bingo cards, prompting players to mark missed numbers and celebrating with winners. Currently, a centralized server is being designed to plan, schedule and monitor the daily activities of the robots. Specific behaviors are planned and performed locally by each individual robot platform.

The integration of planning and scheduling techniques has been investigated over the past several years in such robotic applications as container transportation robots (Alami et al. 1998), office assistant robots (Beetz and Bennewitz 1998), planetary rovers (Estlin et al. 2007), hospital assistant robots (Pecora and Cesta 2002), and eldercare robots (Pineau et al. 2003; Cesta et al. 2011). In these applications, single robot approaches are more commonly studied. With respect to HRI activities, existing work has mainly focused on automated reasoning about the schedule of a single user. For example, the Pearl robot (Pineau et al. 2003) uses the Autominder system (Pollack 2005) to reason about an elderly person's current and planned activities to determine if and when reminders should be provided. The Autominder system has not been extended to consider multiple users. The Cobot robots (Coltin, Veloso, and Ventura 2011) plan and schedule HRI activities, including semi-autonomous telepresence, and office tasks based on requests from several users. However, the planning and scheduling are managed independently and the user schedules are not considered as constraints for the robots' activities. Although multiple user schedules have been considered in other non-robotic scheduling and optimization applications (e.g., building energy conservation (Kwak et al. 2012)), in this work we focus on problems in which an integration of both planning and scheduling is required to reason about the schedules of multiple users, limited resources and metric quantities, and both single- and multi-user HRI activities.

The Problem

We define the main elements of the proposed problem: the environment in which the residents (users) and robots interact, the constraints, the goal and preferences. The constraints for the telepresence and Bingo activities were obtained from meetings with directors, healthcare professionals and residents from Toronto area retirement homes.

The Retirement Home Environment

We consider a floor in a retirement home. The environment consists of rooms, corridors and hallways that are discretized as a set of locations, L (l_1 ... l_n), within which the users and robots will interact. The set of locations and the distance between any two locations ($d_{i,j}$) are known a priori.

Users

The users are the residents of the retirement home. We consider a set of users, U (u_1 ... u_n), for which each user u_k has his/her own *profile*. The profile consists of the user's private *room* location; a minimum, att_min_{uk}, and maximum, att_max_{uk}, number of Bingo games to play in a day; and his/her own distinct *schedule* for the day, representing the user availability (in time and space) for interaction with a robot.

A day for users starts at 7am and ends at 7pm. Within this time frame, users in different locations can be either available or unavailable for interaction with a robot. All users are considered unavailable during breakfast (8am-9am), lunch (12pm-1pm), and dinner (5pm-6pm) and can have other unavailabilities already scheduled.

The Assistive Robots

We consider a set of assistive robots, R (r_1 … r_n), in which each robot r_l is able to execute the following activities: 1) *move* from one discrete location to another at a constant speed v_{rl}, 2) perform a *telepresence* session with a user, 3) perform a *Bingo* session with a group of users, 4) provide a *reminder* to each user prior to a Bingo game, and 5) *recharge* its battery at a charging station. Since battery consumption depends on the activity, whenever the robot, r_l, executes an activity, its *battery level*, bl_{rl}, must remain within bounds (i.e., $bl\_min_{rl} <= bl_{rl} <= bl\_max_{rl}$). A constant rate cr_move_{rl} is used to specify the power consumed for the moving activity (e.g., V/m). Each HRI activity has a different constant consumption rate (e.g., V/min): cr_telep_{rl}, cr_remind_{rl} and cr_bingo_{rl} for the

corresponding activities. Battery power is regained through a charging station. A constant recharging rate rr_{rl} (e.g., V/min) is used to estimate the duration of a recharging process of a robot r_l. The battery of the robot can be recharged up to bl_max_{rl}.

Charging Stations

A set of charging stations, CS (cs_1 … cs_n), exists for recharging. Each station is at a fixed location and can accommodate at most one robot at a time.

Telepresence Sessions

A set of *telepresence sessions*, S (s_1 … s_n), must be scheduled during the day. Each session s_y is characterized by: 1) the user u_k; 2) the duration, dur_{sy}, (e,g, 30 min); and 3) the time window(s) it can occur in. The session should always take place in the user's room (l_{uk}).

Bingo Games

A set of *Bingo games*, G (g_1 … g_n), should be scheduled during the day, if possible. For each game g_z, the robots will assign, find, and remind participants prior to the game and, then, play Bingo at a specific location, the games room (l_{game}), at the scheduled time. Only one game can occur at any given time. Only one robot can conduct the game, but the robots can collaborate to deliver the reminders. Each game g_z is characterized by: 1) the duration of the game, dur_{gz}, (e.g., 60 min) and of the reminder, dur_remind_{gz}; 2) the minimum and maximum number of participants, p_min_{gz} and p_max_{gz}; and 3) the time window(s) in which it can occur.

The group of participating users of a game g_z is not known a priori nor is the time of each game. Users are assigned to each game based on their schedules and attendance preferences and games are scheduled to fit the users' availabilities. Reminders must be delivered to *all* assigned users between 15-120 minutes before the game starts. It is assumed that the users will go to the games room at the time specified.

Robot Activities

We describe below the conditions and constraints of the available robot activities.

Navigate to a target location: the robot has to have enough battery power to reach the target location l_j from its current location l_i. The power consumption and the duration of the moving activity are $d_{i,j} \times cr\_move_{rl}$ and $d_{i,j} / v_{rl}$, respectively.

Recharge battery: the robot has to be in a location with an idle charging station and the battery level has to be less than the battery capacity, $bl_{rl} < bl\_max_{rl}$. The duration of the activity is $(bl\_max_{rl} - bl_{rl})/rr_{rl}$.

Perform Telepresence Session: the robot has to be in the private room of the specified user, who must be available during the entire duration (dur_{sy}) of the activity. The power consumption of the activity is $dur_{sy} \times cr\_telep_{rl}$.

Play Bingo Game: the robot has to be in the games room, no other game can be ongoing, and all users must be available during the entire duration (dur_{gz}) of the game. All assigned users must have been reminded 15-120 minutes before the game starts. The power consumption of the Bingo activity is $dur_{gz} \times cr\_bingo_{rl}$.

Remind User: the robot has to be at the same location as the user, who cannot be interacting with another robot and must be available during the entire duration (dur_remind_{gz}) of the activity. The power consumption of the reminder activity is $dur\_remind_{gz} \times cr\_remind_{rl}$.

In all the activities (except recharging), the robot has to have enough power to reach a location that has a charging station after the activity is completed.

Input, Goal, and Preferences

The *input* of the problem is the sets of locations L, users U (including their corresponding profiles), charging stations CS, available robots R (with their initial location and corresponding velocity, battery levels and limits, and consumption rates), and the requested telepresence sessions S and Bingo games G with their corresponding properties. The *goal* is to have a plan of robot activities in which: 1) all the requested telepresence sessions are scheduled, and 2) the requested Bingo games and reminders are scheduled, if possible, given that user attendance preferences have to be satisfied. All robots must be at a recharging location at the end of the day. As a *multi-objective optimization problem*, we want to: 1) perform as many Bingo games as possible, 2) have as many users playing Bingo as possible, 3) provide reminders as close as possible to the game times, and 4) expend as little battery power as possible.

An Automated P&S Approach

We address the proposed problem using a P&S approach. Herein, we use the itSIMPLE Knowledge Engineering (KE) (Vaquero et al. 2009; 2013) tool that follows an object-oriented modeling approach using the *Unified Modeling Language* (UML) (OMG 2005) and generates a PDDL model of the target problem.

Domain Modeling

A visualization of the modeled object types (classes), fluents and operators is provided in the UML class diagram in Figure 1. The most important classes are: *Location*, *GamesRoom*, *ChargingStation*, *Robot*, *User*, *TelepresenceSession*, *BingoGame* and *Global*. The *Location* and *GamesRoom* (a specialization of Location) represent the topology of the retirement home. The *distance* between locations, and the distance between each available charging station and these locations are represented in the class *Global.* A games room is said to be *free* when no game is taking place at the location. A *ChargingStation* is said to be *idle* when no robot is docked for charging. Moreover, *Robots* and *Users* can only be *at* one location at a time.

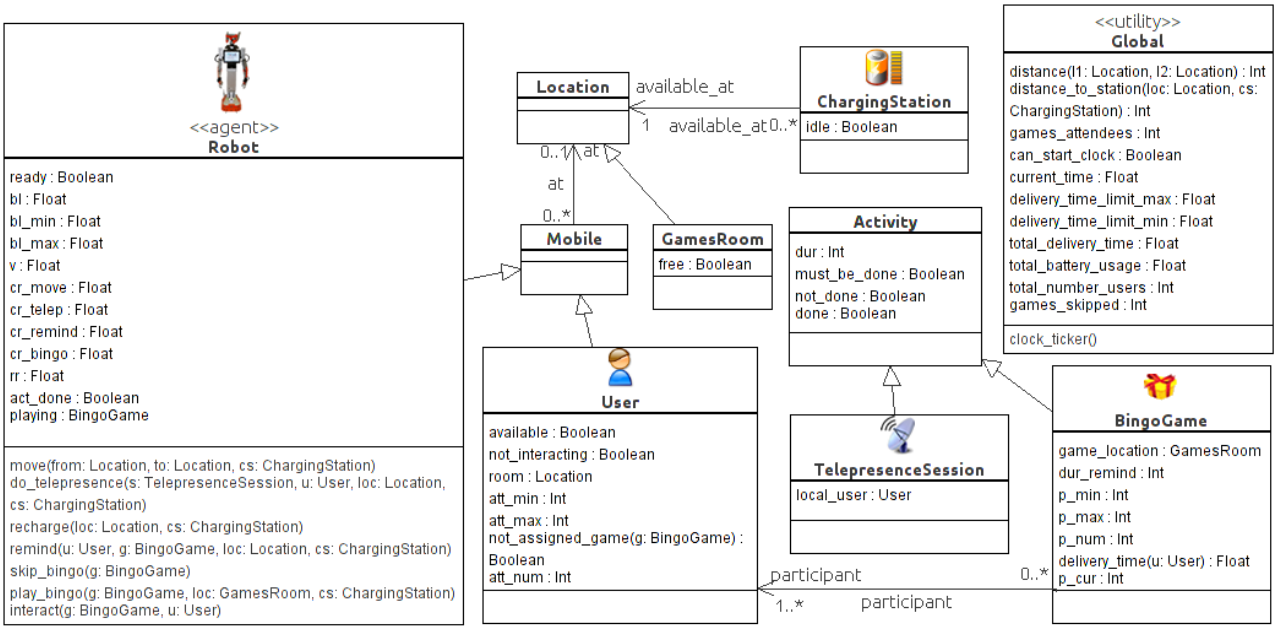


Figure 1. The UML Class diagram of the proposed problem model.

The class *User* has a set of properties to represent the user's profile. The predicate *room* specifies the user's private room while the predicate *available* is used to represent the availability of the user during the day. This availability is translated into PDDL in the form of timed initial literals (TILs) (Edelkamp and Hoffman, 2004) by assigning the *available* predicate to true or false in specific time intervals. We also represent the known locations of the user during the day with TILs. We represent the user preference on attending games (*att_min*, *att_max*), the variable for the number of games attended (*att_num*), and the predicate *not_assigned_game* to list all the games to which a user has not yet been assigned. When a user is interacting with a robot, the predicate *not_interacting* is set to false to prevent other robots from interacting.

The classes *TelepresenceSession* and *BingoGame* represent the HRI activities. Both have the properties: *dur* to represent duration; *not_done* and *done* to represent if the activity has been performed; and *must_be_done*, TILs to represent the time windows in which the activity can be performed. In addition to the properties of the sessions and games introduced in the problem description, we have added the properties *p_num* and *p_cur* to control the number of users reminded by the robots and the number of users playing the game, as well as *delivery_time* to control the time each user is reminded about the game. The difference between the reminder delivery time and the start of the game must be within 15-120 minutes.

Modeling the reminder delivery constraint is not possible without using features that have not been officially incorporated into PDDL. The planner would have to explicitly reason about continuous time during the planning process itself to determine that two actions (*reminder* and *playbingo*) are a certain time apart. This can be done by using PDDL+ which includes *processes* (Fox and Long 2006). Herein, a *process* (called *clock_ticker* in the class *Global*) models an exogenous activity that is triggered for as long as a condition holds (in this case *can_start_clock*), regardless of the action selection process. This mechanism allows us to increment the variable *current_time* in every step of the search, simulating the passage of time. If *current_time* is used in an action's precondition it will hold the exact start time of the action. We use this variable to record the time each user is reminded (*delivery_time*) and also to check if the start time of a game is within the time constraints of the reminders.

The class *Global* also holds global variables including the maximum and minimum time for delivering reminders prior to the games, the total time generated by adding all the lengths of the time intervals between the reminders and the game (*total_delivery_time*), the total amount of battery power consumed by all robots (*total_battery_usage*), the total number of games not played (*game_skipped*), and the number of target users (*total_number_users*). These variables are used to specify the cost function and are manipulated in the specification of the robot actions.

The class *Robot* has all the properties described in the problem description (e.g., velocity, battery level, etc). In addition, we have the predicates *ready*, *act_done*, and *playing*. A robot is *ready* when it is not engaged in any activity and it is *playing* when it is performing a Bingo activity. The predicate *act_done* prevents a robot from

going to a location and performing no action: a robot can only move to another location if it has completed an activity in its current location. As shown in Figure 1, a robot has the following operators: *move* to a target location; *recharge* its battery; *remind* users; *do_telepresence* with a user; *play_bingo* and *interact* with a player during the game; and *skip_bingo* which removes the game from the request list.

In the *reminder* operator, the user is set as a *participant* of the game. In order to play a game after the reminders, a robot has to first start the *play_bingo* action, then it has to perform, in parallel (a required concurrency), the action *interact* with each participant. The *play_bingo* action can only finish when the robot has performed the *interact* action with all assigned players.

In the goal state all sessions and games must be *done* (Bingo games can be either performed or skipped) and the user preferences on game attendance must be satisfied. We aim to minimize the following weighted cost function *f*:

$$f = 500\times(games\_skipped) + 1000\times(total\_number\_users - games\_attendees) + total\_battery\_usage + total\_delivery\_time \quad (1)$$

where the weights are used to express preference on optimizing the number of games and players. Due to space limitations, we present the PDDL code for the proposed model at:

https://docs.google.com/file/d/0B3t9fqfsJqrlTGpndVNxMEdSSlU/edit?pli=1.

Model Variations

The resulting PDDL model includes features that are challenging for most existing planners: metric quantities, optimization, temporal actions, timed initial literals, concurrent actions, and processes. In particular, few planners can properly handle the required concurrency (*R*) and processes (*P*). Therefore, we decided to define model variations to investigate the performance of existing temporal planners. Model ***RP*** is our full model as described above. Model ***RN*** does not consider the reminder time constraint and therefore, does not use processes. Model ***NP*** is our full model without the required concurrency in the Bingo activity. We replace both operators *play_bingo* and *interact* with operators *play_bingo3* and *play_bingo4*, each representing a game activity with a specific number of participants. Representing an operator for each possible number of participants, in this case from 3 to 10, is impractical due to the large number of parameters and, consequently, an exponentially increasing number of action instantiations during the planning procedure. Therefore, the maximum number of Bingo game participants is 4 when using the operators *play_bingo3* and *play_bingo4*. Model ***NN*** is the full model with both required concurrency and processes removed.

Experiments

We chose five planners to investigate: COLIN (Coles et al. 2012), LPG-td (Gerevini, Saetti, and Serina 2004), OPTIC (Benton, Coles, and Coles 2012), POPF (Coles et al. 2010) and SGPlan (Hsu and Wah 2008). All these planners can potentially handle metric quantities, optimization, temporal actions, and timed initial literals. However, only OPTIC, COLIN and POPF handle the required concurrency and processes.

We consider a realistic retirement home environment in which residents have several activities in different locations (e.g., TV room, private room, garden, dining hall) during a day. We assume that each user has a number of 1-hour activities (e.g., physiotherapy, doctor's appointment, family visit, nap), in addition to the meal times, during which the robots cannot disturb him/her (herein called *non-interruptible activities*). Other activities (e.g., walk in the garden, reading in a common area) allow robot interactions (*interruptible activities*); at least one interruptible activity is assumed for each user. We analyze the selected planners for five full-day scenarios in this environment (7am-7pm) – see Table 1. For each full-day scenario, we analyze different numbers of non-interruptible activities for the users. We investigate non-interruptible activity density, defined as *Density k*, (*k = 0,1,2,3,4*), where *k* is the number of non-interruptible activities, in addition to the meals, that each user has per day. The different densities in particular are aimed to study the impact of the user availability constraints on the performance for the selected planners.

Table 1. The number of objects in the five scenarios.

Scenario	*Users*	*Robots*	*Telepresence*	*Bingo*
1	5	2	2	1
2	10	2	4	3
3	15	3	6	5
4	20	3	8	6
5	25	4	10	8

In all scenarios, the telepresence sessions and Bingo games are 30 and 60 minutes long, respectively with time windows from 7am-7pm. Reminders are 2 minutes long. Each game has a minimum of three participants and a maximum of ten participants (in models *NP* and *NN* the maximum is four as previously noted). Every user is willing to attend at most one Bingo game during the day (i.e., *att_min* = 0, *att_max* = 1). Each scenario was designed so that it is feasible to schedule at least one game with five participants. All robots have the following property values, estimated based on the H20 robot platform: *bl_min* = 0, *bl* = *bl_max* = 20, *v* = 20m/min, *rr* = 0.5, *cr_move* = 0.04 and *cr_telep* = *cr_remind* = *cr_bingo* = 0.1.

We run the planners for each model variation with each of the five scenarios and each density on a 64-bit Ubuntu Linux machine with 32 GB of memory. A 1-hour timeout

was used for each planner in each scenario. We measure the solvability of the planners, the runtime, the number of states evaluated, and the number of users attending a game. Table 2 shows the number of scenarios (out of 5) for which each planner was able to generate at least one solution.

Table 2. Number of scenarios solved by each planner. The '-' indicates that the planner could not represent the model, while the '$^{(inv)}$' indicates that the planner generates invalid solutions for some scenarios. Such solutions are not included in the number of scenarios solved.

Planners	**Models**			
	RP	*RN*	*NP*	*NN*
Density 0				
COLIN	$0^{(inv)}$	$0^{(inv)}$	$0^{(inv)}$	$0^{(inv)}$
LPG-td	-	-	-	0
OPTIC	5	5	2	2
POPF	0	$1^{(inv)}$	0	$1^{(inv)}$
SGPlan	-	-	-	0
Density 1				
COLIN	3	3	2	2
LPG-td	-	-	-	0
OPTIC	5	5	2	2
POPF	0	5	0	2
SGPlan	-	-	-	0
Density 2				
COLIN	4	4	2	2
LPG-td	-	-	-	0
OPTIC	5	5	2	2
POPF	0	5	0	2
SGPlan	-	-	-	0
Density 3				
COLIN	4	4	2	2
LPG-td	-	-	-	0
OPTIC	5	5	2	2
POPF	0	5	0	2
SGPlan	-	-	-	0
Density 4				
COLIN	4	4	2	2
LPG-td	-	-	-	0
OPTIC	5	5	2	2
POPF	0	5	0	2
SGPlan	-	-	-	0

As shown in Table 2, the majority of scenarios were solved by some of the planners with models *RP* and *RN* while few scenarios were solved with models *NP* and *NN*. The OPTIC planner was the only P&S system able to solve scenarios with all models and in all investigated densities. Across all investigated models and densities, the solutions generated by the planners for a given scenario varied with respect to the number of robots used, total battery usage, and makespan. For example, COLIN generated plans with only one robot more often than POPF and OPTIC did for *Scenarios 1* and *2*. Having less robots resulted in a lower cost, however, using multiple robots had lower makespan. The number of Bingo games scheduled also varied. The majority of the solutions did not schedule a game at all and had the following common structure for the plan: skip the Bingo games and schedule the assigned robots to implement the requested telepresence sessions, while recharging the robots when necessary; and at the end of the day, the robots assigned in the plan return to the charging station. The solutions with scheduled Bingo games had a similar structure as those with no games, however in these cases the robots assigned in the plan also scheduled reminders to users prior to the start of a Bingo game as well as the game playing session.

LPG-td and SGPlan were not able to solve any of the problem instances with model *NN*, the only model that these planners could represent. We suspect that this is due to the large number of TILs used to represent the user schedules. The COLIN planner was able to generate solutions for the four proposed models and POPF was able to generate solutions only for models *RN* and *NN*. However, none of the solutions from COLIN and POPF for the five scenarios had any Bingo games: all the games were skipped. Furthermore, these two planners generated invalid solutions in problem instances with *Density 0*, for example, scheduling telepresence activities during the breakfast period. Interestingly, neither planner generated invalid solutions at higher densities. This issue occurs when there is no non-interruptible activity in beginning of the day (7am), i.e. all users start with the variable *available* set to true in the initial state and this variable does not change until the beginning of breakfast, when it is set to false. In the problem instances with density greater than zero, some users have non-interruptible activities starting at 7am, so their corresponding variable *available* is false in the initial state. In such cases no invalid solutions were generated. With COLIN we observed that the issue seems to be related to the compression-safe action detection mechanism (Coles et al. 2012). When this mechanism is disabled, the issue no longer occurs. POPF has a similar mechanism; however, disabling it does not eliminate the issue. Given that the compression-safe action detection is a default mechanism in both planners, we decided to keep it enabled in our experiments. Further experimentation and analysis is needed.

Table 3 shows the runtime and number of states evaluated for COLIN and POPF to find a solution with the four models. In most cases, the planners stopped before the timeout. The density of non-interruptible user activities seems to have some impact on the performance of both planners. Different trends are observed in Table 3. For example, the runtime decreases as the density increases in *Scenario 3* for COLIN with models *RP* and *RN*. Moreover, the runtime increases in *Scenario 5* for POPF with model RN as the density increases.

As OPTIC had the best performance, we ran it to search for better solutions until the timeout. OPTIC was the only planner that was able to find solutions that included Bingo games. Table 4 shows the runtime, number of states evaluated and the number of users playing Bingo games in the plans found by the OPTIC planner. This table focuses

on the *first* and *last* solutions found to illustrate how fast the planner can find a solution and the quality of the best solution found. For problems for which the planner generated no solution or only one solution, we show the time the planner stopped instead.

As shown in Table 4, plans with Bingo games were only found in *Scenario 1*. Most of these plans were found with models *RN* and *NN*, i.e., the models without the reminder time constraint. Problem instances from *Scenario 1* with *Density 0* are the only instances in which OPTIC generated plans with Bingo games with all models. OPTIC generated solutions for *Scenarios 3*, *4* and *5* across all non-interruptible user activity densities only with models *RP* and *RN* (models with required concurrency). Most of the solutions with the highest number of Bingo participants were generated with model *NN*, the simplest PDDL model. During the optimization process of all the scenarios, most improvements to the plan resulted in lower battery consumption. OPTIC stopped running before the timeout in most cases. While the reason is unclear but we suspect that it reached its internal memory limits.

With respect to the impact of the different non-interruptible user activity densities, Table 4 shows that both the runtime and the number of evaluated states increased as the density was increased with model *RP* and *RN* in *Scenarios 4* and *5*. An increase in runtime can also be observed with modes *NP* and *NN* in *Scenario 2*. In order to investigate whether the different non-interruptible user activity densities affected the performance of the planner on finding a solution with a Bingo game, we further investigated the very first solutions found by OPTIC in which a Bingo game was scheduled. Table 5 shows the runtime and number of evaluated states for those cases in *Scenario 1* across the four models and the five densities. The density increment tends to decrease the runtime to find a solution with a Bingo game as well as the number of evaluated states with models *RN* and *NP*. We suspect that this pattern is due to the decreasing number of time windows in which a game can be scheduled leading to a reduction in the alternatives during the search.

Discussion

The experimental results show that existing domain-independent temporal planners are not able to solve the proposed multi-robot, multi-user, single and multi-user HRI activities problem for realistic scenarios. Although some of the planners provide feasible solutions, optimal solutions do not appear to be achievable. In particular, the expected optimization of the number of Bingo players is observed in very few small-scale cases, most of the time in models that do not consider the full requirements of the problem.

The advancement of P&S technology in representing and solving problems with temporal constraints, time-windows and numeric quantities is noticeable since the 1980s (Boddy, Cesta, and Smith 2004). However, the challenges in modeling and solving problems that require integrated P&S with such a combination of complex features are evident from our experiment. From the modeling perspective, not all temporal requirements can be represented in standard PDDL. While few planners can handle the aforementioned features together, even fewer represent the PDDL+ features our problem requires. Due to the few temporal planners available for these challenging problems, the modeling process becomes driven by the planner at hand. Namely, we found ourselves faced with tailoring the model to the solver's abilities at the expense of accurately representing our problem.

From the perspective of a user of AI planning technology (e.g., a roboticist who wants to focus on the challenges of sensing, navigation, and HRI), current domain-independent AI planning technology is not up to the task. We hope that the application we have introduced can form a challenge to spur advances in this direction.

We intend to extend this work to investigate timeline-based planning (Muscettola 1994) and scheduling approaches such as constraint programming and mixed-integer programming. Our preliminary indication is that none of these technologies will be able to reliably solve these problems. If that is the case, we intend to identify the most promising technology and investigate its extension to be able to solve our real-world problem.

Conclusion

We have introduced a new planning and scheduling problem in which multiple robots have to interact with residents in a retirement home environment to perform single- and multi-user activities while considering the users' schedules. We have investigated an AI P&S approach by: 1) designing variations of a PDDL model and realistic problem instances using the itSIMPLE KE tool, and 2) studying the performance of five state-of-the-art domain-independent temporal planners. Experimental results demonstrate that current temporal planners can sometimes provide valid solutions even with a complex combination of model features. However, in most of the cases they failed to provide solutions in which both single- and multi-user activities are present, even when using simplified models. The results reinforce the existing technology gap in the AI P&S approach for both modeling and solving real problems that combine temporal, numeric and optimization requirements.

Acknowledgments

This research has been funded by a Natural Sciences and Engineering Research Council of Canada (NSERC) Collaborative Research and Development Grant and by Dr Robot Inc.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An Architecture for Autonomy. *International Journal of Robotics Research*, 17: 315-337.

Beetz, M., and Bennewitz, M. 1998. Planning, Scheduling, and Plan Execution for Autonomous Robot Office Couriers, *Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, Workshop Notes, 1998.

Benton, J.; Coles, A. J.; and Coles, A. I. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of the International Conference on Automated Planning and Scheduling* (ICAPS-12), pp. 2-10.

Boddy, M.; Cesta, A.; and Smith, S. ICAPS-04 Workshop on Integrating Planning into Scheduling. Workshop proceeding: http://pst.istc.cnr.it/wipis-at-icaps-04/WIPIS-ICAPS04-Notes.pdf

Cesta, A.; Cortellessa, G.; Rasconi, R.; Pecora, F.; Scopelliti, M.; and Tiberio, L. 2011. Monitoring Older People with the RoboCare Domestic Environment: Interaction Synthesis and User Evaluation. *Computational Intelligence*, 27(1): 60–82.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling* (ICAPS-10), pp. 1-8.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. COLIN: Planning with Continuous Linear Numeric Change. *Journal of Artificial Intelligence Research*. 44: 1-96.

Coltin, B.; Veloso, M.; and Ventura, R. 2011. Dynamic User Task Scheduling for Mobile Robots. In *Proceedings of the AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots*, pp. 1-6.

Dr Robot. 2014. H20 Wireless Networked Autonomous Humanoid Mobile Robot. Dr Robot Inc. http://www.drrobot.com/products_h20.asp.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th International Planning Competition. Tech. rep. 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Estlin, T.; Gaines, D.; Chouinard, C.; Castano, R.; Bornstein, B.; Judd, M.; Nesnas, I.; and Anderson, R. 2007. Increased Mars Rover Autonomy using AI Planning, Scheduling and Execution. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 4911-4918.

Fasola, J., and Mataric, M. 2012. Using Socially Assistive Human-Robot Interaction to Motivate Physical Exercise for Older Adults. *IEEE, Special Issue on Quality of Life Technology*, T. Kanade, ed., 100(8): 2512-2526.

Fox, M. 1994. ISIS: A Retrospective. In Zweben, M., and Fox, M. 1994. *Intelligent Scheduling*. Morgan Kaufmann, pp. 3–28.

Smith, D. E.; Frank, J.; and Jonsson, A. K. 2000. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review*, 15(1): 1-34.

Fox, M., and Long, D. 2006. Modelling Mixed Discrete Continuous Domains for Planning. *Journal of Artificial Intelligence Research* 27:235–297.

Gerevini, A.; Saetti, A.; and Serina, I. 2004. LPG-TD: a Fully Automated Planner for PDDL2.2 Domains. In *Proceedings of the International Conference on Automated Planning and Scheduling* (ICAPS-04), booklet of the system demo section..

Hsu, C., and Wah, B. W. 2008. The SGPlan Planning System in IPC-6. In the booklet of the International Planning Competition (IPC), International Conference on Planning and Scheduling.

Kidd, C. D.; Taggart, W.; and Turkle, S. 2006. A Sociable Robot to Encourage Social Interaction among the Elderly, In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 3972-3976.

Kwak, J.; Varakantham, P.; Maheswaran, R.; Tambe, M.; Jazizadeh, F.; Kavulya, G.; Klein, L.; Becerik-Gerber, B.; Hayes, T.; and Wood, W. 2012. SAVES: a sustainable multiagent application to conserve building energy considering occupants. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, v.1, pp. 21-28.

Louie, W. G.; Han, R.; and Nejat, G. 2013. A Socially Assistive Robot to Promote Stimulating Recreational Activities at Long-term Care Facilities, *Journal of Medical Devices-Transactions of the ASME*, 7, 030944-1.

Louie, W. G.; Vaquero, T.; Nejat, G.; and Beck, J. C. 2014 An Autonomous Assistive Robot for Planning, Scheduling and Facilitating Multi-User Activities, In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

McColl, D.; Louie, W. G.; and Nejat, G. 2013. Brian 2.1: A Socially Assistive Robot for the Elderly and Cognitively Impaired. *IEEE Robotics & Automation Magazine*, 20(1): 74-83.

Montemerlo, M.; Prieau, J.; Thrun, S.; and Varma, V. 2002. Experiences with a mobile robotics guide for the elderly. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 587–592.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. *Intelligent Scheduling*, ed., Zweben, M. and Fox, M.S., M. Kauffmann.

OMG. 2005. OMG Unified Modeling Language Specification, Version 2.0.

Pecora, F., and Cesta, A. 2002. Planning and Scheduling Ingredients for a Multi-Agent System. In *Proceedings for the of UK PLANSIG'02 Workshop*, pp. 135-148.

Pineau, J.; Montemerlo, M.; Pollack, M.; Roy, N.; and Thrun, S. 2003. Towards Robotic Assistants in Nursing Homes: Challenges and Results. *Robotics and Autonomous Systems*, 42(3-4): 271-281.

Pollack, M. 2005. Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment. *AI Magazine*, 26(2): 9–24.

Vaquero, T.; Silva, J. R.; Ferreira, M.; Tonidandel, F.; and Beck, J. C. 2009. From requirements and analysis to PDDL in itSIMPLE3.0. In *Proceedings of the International Competition on Knowledge Engineering for Planning and Scheduling* (ICKEPS), at ICAPS 2009, Thessaloniki, Greece, pp. 1-8.

Vaquero, T.; Silva, J. R.; Tonidandel, F.; and Beck, J. C. 2013. itSIMPLE: Towards an Integrated Design System for Real Planning Applications. *The Knowledge Engineering Review Journal*, 28(2): 215–230.

Table 3. Runtime (s) and number of evaluated states for the planners COLIN and POPF. The '-' indicates that no solution was found and the '(inv)' indicates the invalid solutions.

	POPF								**COLIN**							
	RP		*RN*		*NP*		*NN*		*RP*		*RN*		*NP*		*NN*	
Scenarios	runtime (s)	states	runtime (s)	states	runtime (s)	states	runtime (s)	states	runtime (s)	states	runtime (s)	states	runtime (s)	states	runtime (s)	states
Density 0																
1	0.04	-	0.04	11	0. 06	-	0.10	11	0.04(inv)	8	0.06(inv)	8	0.16(inv)	8	0.10(inv)	8
2	0.07	-	0.18(inv)	30	3.12	-	312.7(inv)	30	0.14(inv)	16	0.14(inv)	16	1806.6(inv)	16	179.0(inv)	16
3	0.17	-	1.00(inv)	65	46.9	-	timeout	-	0.70(inv)	30	0.64(inv)	30	timeout	-	timeout	-
4	0.32	-	2.56(inv)	107	97.6	-	96.5	-	1.70(inv)	56	1.52(inv)	56	121.4	-	120.8	-
5	0.74	-	10.0(inv)	188	308.3	-	282.8	-	5.82(inv)	59	4.92(inv)	59	432.6	-	414.5	-
Density 1																
1	0.04	-	0.12	127	0.12	-	0.24	127	0.08	127	0.08	127	0.20	127	0.14	127
2	0.08	-	0.66	336	3.20	-	318.6	336	1.88	2326	1.74	2326	1890.0	2326	194.1	2326
3	0.18	-	4.60	933	46.8	-	timeout	-	906.2	199559	628.3	199559	timeout	-	timeout	-
4	0.32	-	15.2	1676	98.1	-	96.3	-	2838.7	-	2118.1	-	121.7	-	120.7	-
5	0.74	-	66.4	3164	307.9	-	284.7	-	timeout	-	timeout	-	433.2	-	416.5	-
Density 2																
1	0.05	-	0.14	139	0.14	-	0.24	139	0.08	139	0.08	139	0.22	139	0.16	139
2	0.08	-	0.72	359	3.22	-	344.4	359	0.94	1052	0.86	1052	1905.8	1052	203.9	1052
3	0.18	-	6.08	1264	46.9	-	timeout	-	111.5	27155	76.9	27155	timeout	-	timeout	-
4	0.32	-	14.1	1615	98.4	-	95.9	-	2362.9	255989	1493.7	255989	122.1	-	120.5	-
5	0.75	-	85.7	4060	307.2	-	285.0	-	timeout	-	timeout	-	432.0	-	414.0	-
Density 3																
1	0.05	-	0.14	139	0.14	-	0.28	139	0.08	139	0.08	139	0.22	139	0.16	139
2	0.08	-	0.70	359	3.26	-	324.2	359	0.96	1052	0.88	1052	2000.7	1052	158.5	1052
3	0.18	-	5.96	1228	46.9	-	timeout	-	57.2	13911	39.0	13911	timeout	-	timeout	-
4	0.32	-	13.5	1570	97.7	-	96.0	-	2405.8	256743	1548.6	256743	120.8	-	120.7	-
5	0.76	-	126.9	5375	310.9	-	283.8	-	timeout	-	timeout	-	431.9	-	414.9	-
Density 4																
1	0.06	-	0.14	139	0.14	-	0.24	139	0.08	139	0.08	139	0.22	139	0.16	139
2	0.09	-	0.72	359	3.24	-	327.2	359	0.96	1052	0.86	1052	2043.0	1052	173.6	1052
3	0.20	-	6.00	1228	47.0	-	timeout	-	55.5	13911	39.4	13911	timeout	-	timeout	-
4	0.34	-	19.9	2080	98.1	-	95.9	-	1711.9	191594	1119.9	191594	121.0	-	120.3	-
5	0.76	-	313.4	11352	308.2	-	284.6	-	timeout	-	timeout	-	432.6	-	413.0	-

Table 4. OPTIC planner performance in all models and scenarios: runtime (s), number of states evaluated and the number of Bingo game participants (part.) for the first and the last solutions found by the planner. The '*' indicates that the planner stopped at the specified time and '-' that no solution was found.

Scen-arios	*RP*						*RN*						*NP*						*NN*					
	first			last			first			last			first			last			first			last		
	runtime (s)	states	part.	runtime (s)	states	part.	runtime (s)	states	part.	runtime (s)	states	part.	runtime (s)	states	part.	runtime (s)	states	part.	runtime (s)	states	part.	runtime (s)	states	part.
Density 0																								
1	0.06	11	0	389.1	67419	3	0.06	11	0	730.6	92553	3	0.18	11	0	3197.0	94794	4	0.14	11	0	2179.6	188659	4
2	0.34	40	0	821.5	89927	0	0.32	40	0	745.2	89927	0	273.5	40	0	1772.5	6585	0	127.8	40	0	1612.9	6585	0
3	27.9	2268	0	260.0	14516	0	22.6	2268	0	224.8	14516	0	timeout	-	-	-	-	-	timeout	-	-	-	-	-
4	48.5	2508	0	1271.0	41283	0	35.2	2508	0	1062.0	41283	0	141.2*	-	-	-	-	-	140.1*	-	-	-	-	-
5	345.1	7692	0	2561.5*	-	-	264.7	7692	0	2987.5*	-	-	485.3*	-	-	-	-	-	458.5*	-	-	-	-	-
Density 1																								
1	0.06	8	0	2002.2	90623	3	0.06	8	0	1775.7	183257	3	0.20	8	0	2936.0*	-	-	0.14	8	0	1478.0	167930	4
2	0.24	26	0	190.0	22792	0	0.24	26	0	179.3	22792	0	254.3	26	0	2761.1	11713	0	128.5	26	0	2505.7	11713	0
3	26.7	2115	0	1163.6	59770	0	21.8	2115	0	1040.1	59770	0	timeout	-	-	-	-	-	timeout	-	-	-	-	-
4	34.4	1882	0	2027.2*	-	-	24.5	1882	0	2358.0*	-	-	140.5*	-	-	-	-	-	140.1*	-	-	-	-	-
5	158.8	4062	0	2593.4*	-	-	118.2	4062	0	2995.6*	-	-	487.3*	-	-	-	-	-	463.2*	-	-	-	-	-
Density 2																								
1	0.06	8	0	timeout	-	-	0.06	8	0	1269.6	154476	3	0.18	8	0	3529.2*	-	-	0.14	8	0	1250.8	139669	4
2	0.24	26	0	1209.9	151592	0	0.22	26	0	1115.7	151592	0	256.1	26	0	2879.9	11713	0	136.8	26	0	2657.7	11713	0
3	26.2	2129	0	1604.2	86052	0	21.3	2129	0	1403.1	86052	0	timeout	-	-	-	-	-	timeout	-	-	-	-	-
4	58.7	2797	0	1981.9*	-	-	44.0	2797	0	2287.8*	-	-	141.8*	-	-	-	-	-	139.7*	-	-	-	-	-
5	316.6	6977	0	2587.9*	-	-	234.8	6977	0	2955.9*	-	-	487.4*	-	-	-	-	-	464.2*	-	-	-	-	-
Density 3																								
1	0.06	8	0	timeout	-	-	0.06	8	0	1945.2	202444	4	0.20	8	0	timeout	-	-	0.14	8	0	2764.9	281129	4
2	0.26	26	0	1272.9	151592	0	0.22	26	0	1133.7	151592	0	354.0	26	0	2993.6	11713	0	207.3	26	0	2811.5	11713	0
3	26.8	2075	0	965.9	50272	0	20.7	2075	0	1609.5	100868	0	timeout	-	-	-	-	-	timeout	-	-	-	-	-
4	63.5	2959	0	1990.5*	-	-	48.5	2959	0	2292.8*	-	-	140.7*	-	-	-	-	-	140.3*	-	-	-	-	-
5	331.7	7338	0	2541.6*	-	-	253.8	7338	0	2967.7*	-	-	485.3*	-	-	-	-	-	468.1*	-	-	-	-	-
Density 4																								
1	0.06	8	0	timeout	-	-	0.06	8	0	1974.3	222945	3	0.22	8	0	1037.5	90968	3	0.12	8	0	1315.0	172596	4
2	0.24	26	0	1115.1	143131	0	0.22	26	0	1031.5	143131	0	807.0	26	0	2707.3	11360	0	581.6	26	0	timeout	-	-
3	24.6	2075	0	898.7	50272	0	20.1	2075	0	1559.2	100868	0	timeout	-	-	-	-	-	timeout	-	-	-	-	-
4	66.4	3047	0	1537.6	50611	0	51.9	3047	0	1311.3	50611	0	141.2*	-	-	-	-	-	139.8*	-	-	-	-	-
5	461.8	9850	0	1776.2	32746	0	346.8	9850	0	1377.4	32746	0	486.0*	-	-	-	-	-	461.5*	-	-	-	-	-

Table 5. Runtime (s) and number of evaluated states for the planner OPTIC to find a solution with a Bingo game in the *Scenario 1*. The '-' indicates that the planner could not find a solution with a Bingo game.

Density	**Models**							
	RP		*RN*		*NP*		*NN*	
	runtime (s)	states	runtime (s)	states	runtime (s)	states	runtime (s)	states
0	389.1	67419	305.3	67547	1430.5	84353	338.9	56550
1	562.7	73856	271.7	67234	-	-	314.2	56316
2	-	-	269.7	66577	-	-	310.8	55659
3	-	-	263.5	64858	-	-	304.7	54286
4	-	-	296.2	71191	1037.5	90968	284.1	54212

Planning for Decentralized Control of Multiple Robots Under Uncertainty

Christopher Amato[1], **George D. Konidaris**[1], **Gabriel Cruz**[1]

Christopher A. Maynor[2], **Jonathan P. How**[2] and **Leslie P. Kaelbling**[1]

[1]CSAIL, [2]LIDS
MIT
Cambridge, MA 02139

Abstract

We describe a probabilistic framework for synthesizing control policies for general multi-robot systems, given environment and sensor models and a cost function. Decentralized, partially observable Markov decision processes (Dec-POMDPs) are a general model of decision processes where a team of agents must cooperate to optimize some objective (specified by a shared reward or cost function) in the presence of uncertainty, but where communication limitations mean that the agents cannot share their state, so execution must proceed in a decentralized fashion. While Dec-POMDPs are typically intractable to solve for real-world problems, recent research on the use of macro-actions in Dec-POMDPs has significantly increased the size of problem that can be practically solved as a Dec-POMDP. We describe this general model, and show how, in contrast to most existing methods that are specialized to a particular problem class, it can synthesize control policies that use whatever opportunities for coordination are present in the problem, while balancing off uncertainty in outcomes, sensor information, and information about other agents. We use three variations on a warehouse task to show that a single planner of this type can generate cooperative behavior using task allocation, direct communication, and signaling, as appropriate.

Introduction

The decreasing cost and increasing sophistication of recently available robot hardware has the potential to create many new opportunities for applications where teams of relatively cheap robots can be deployed to solve real-world problems. Practical methods for coordinating such multi-robot teams are therefore becoming critical. A wide range of approaches have been developed for solving specific classes of multi-robot problems, such as task allocation [15], navigation in a formation [5], cooperative transport of an object [20], coordination with signaling [6] or communication under various limitations [33]. Broadly speaking, the current state of the art in multi-robot research is to hand-design special-purpose controllers that are explicitly designed to exploit some property of the environment or produce a specific desirable behavior. Just as in the single-robot case, it would be much more desirable to instead specify a world model and a cost metric, and then have a general-purpose planner automatically derive a controller that minimizes cost, while remaining robust to the uncertainty that is fundamental to real robot systems [37].

The decentralized partially observable Markov decision process (Dec-POMDP) is a general framework for representing multiagent coordination problems. Dec-POMDPs have been studied in fields such as control [1, 23], operations research [8] and artificial intelligence [29]. Like the MDP [31] and POMDP [17] models that it extends, the Dec-POMDP model is very general, considering uncertainty in outcomes, sensors and information about the other agents, and aims to optimize policies against a a general cost function. Dec-POMDP problems are often characterized by incomplete or partial information about the environment and the state of other agents due to limited, costly or unavailable communication. Any problem where multiple agents share a single overall reward or cost function can be formalized as a Dec-POMDP, which means a good Dec-POMDP solver would allow us to automatically generate control policies (including policies over when and what to communicate) for very rich decentralized control problems, in the presence of uncertainty. Unfortunately, this generality comes at a cost: Dec-POMDPs are typically infeasible to solve except for very small problems [3].

One reason for the intractability of solving large Dec-POMDPs is that current approaches model problems at a low level of granularity, where each agent's actions are primitive operations lasting exactly one time step. Recent research has addressed the more realistic *MacDec-POMDP* case where each agent has *macro-actions*: temporally extended actions which may require different amounts of time to execute [3]. MacDec-POMDPs cannot be reduced to Dec-POMDPs due to the asynchronous nature of decision-making in this context — some agents may be choosing new macro-actions while others are still executing theirs. This enables systems to be modeled so that coordination decisions only occur at the level of deciding which macro-actions to execute. MacDec-POMDPs retain the ability to coordinate agents while allowing near-optimal solutions to be generated for significantly larger problems than would be possible using other Dec-POMDP-based methods.

Macro-actions are a natural model for the modular controllers often sequenced to obtain robot behavior. The macro-action approach leverages expert-designed or learned controllers for solving subproblems (e.g., navigating to a waypoint or grasping an object), bridging the gap between

traditional robotics research and work on Dec-POMDPs. This approach has the potential to produce high-quality general solutions for real-world heterogeneous multi-robot coordination problems by automatically generating control and communication policies, given a model.

The goal of this paper is to present this general framework for solving decentralized cooperative partially observable robotics problems and provide the first demonstration of such a method running on real robots. We begin by formally describing the Dec-POMDP model, its solution and relevant properties, and describe MacDec-POMDPs and a memory-bounded algorithm for solving them. We then describe a process for converting a robot domain into a MacDec-POMDP model, solving it, and using the solution to produce a SMACH [9] finite-state machine task controller. Finally, we use three variants of a warehouse task to show that a MacDec-POMDP planner allows coordination behaviors to emerge automatically by optimizing the available macro-actions (allocating tasks, using direct communication, and employing signaling, as appropriate). We believe the MacDec-POMDP represents a foundational algorithmic framework for generating solutions for a wide range of multi-robot systems.

Decentralized, Partially Observable Markov Decision Processes

Dec-POMDPs [8] generalize partially observable Markov decision processes to the multiagent, decentralized setting. Multiple agents operate under uncertainty based on (possibly different) partial views of the world, with execution unfolding over a bounded or unbounded sequence of steps. At each step, every agent chooses an action (in parallel) based purely on locally observable information, resulting in an immediate reward and an observation being obtained by each individual agent. The agents share a single reward or cost function, so they should cooperate to solve the task, but their local views mean that operation is decentralized during execution.

As depicted in Fig. 1, a Dec-POMDP [8] involves multiple agents that operate under uncertainty based on different streams of observations. We focus on solving sequential decision-making problems with discrete time steps and stochastic models with finite states, actions, and observations, though the model can be extended to continuous problems. A key assumption is that state transitions are *Markovian*, meaning that the state at time t depends only on the state and events at time $t-1$. The reward is typically only used as a way to specify the objective of the problem and is not observed during execution.

More formally, a Dec-POMDP is described by a tuple $\langle I, S, \{A_i\}, T, R, \{\Omega_i\}, O, h\rangle$, where

- I is a finite set of agents.
- S is a finite set of states with designated initial state distribution b_0.
- A_i is a finite set of actions for each agent i with $A = \times_i A_i$ the set of joint actions, where $\times$ is the Cartesian product operator.

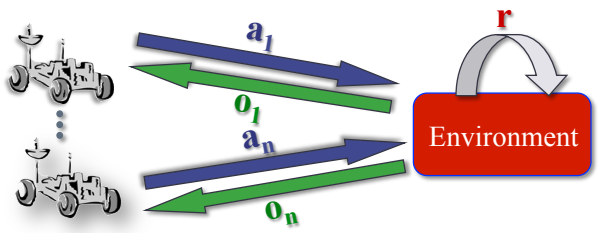


Figure 1: Representation of n agents in a Dec-POMDP setting with actions a_i and observations o_i for each agent i along with a single reward r.

- T is a state transition probability function, $T : S \times A \times S \rightarrow [0, 1]$, that specifies the probability of transitioning from state $s \in S$ to $s' \in S$ when the actions $\vec{a} \in A$ are taken by the agents. Hence, $T(s, \vec{a}, s') = \Pr(s'|\vec{a}, s)$.
- R is a reward function: $R : S \times A \rightarrow \mathbb{R}$, the immediate reward for being in state $s \in S$ and taking the actions $\vec{a} \in A$.
- Ω_i is a finite set of observations for each agent, i, with $\Omega = \times_i \Omega_i$ the set of joint observations.
- O is an observation probability function: $O : \Omega \times A \times S \rightarrow [0, 1]$, the probability of seeing observations $\vec{o} \in \Omega$ given actions $\vec{a} \in A$ were taken which results in state $s' \in S$. Hence $O(\vec{o}, \vec{a}, s') = \Pr(\vec{o}|\vec{a}, s')$.
- h is the number of steps until the problem terminates, called the horizon.

Note that while the actions and observation are factored, the state need not be. This flat state representation allows more general state spaces with arbitrary state information outside of an agent (such as target information or environmental conditions). Because the full state is not directly observed, it may be beneficial for each agent to remember a history of its observations. Specifically, we can consider an action-observation history for agent i as

$$H_i^A = (s_i^0, a_i^1, \ldots, s_i^{l-1}, a_i^l).$$

Unlike in POMDPs, it is not typically possible to calculate a centralized estimate of the system state from the observation history of a single agent, because the system state depends on the behavior of all of the agents.

Solutions

A solution to a Dec-POMDP is a *joint policy*—a set of policies, one for each agent in the problem. Since each policy is a function of history, rather than of a directly observed state, it is typically represented as either a policy tree, where the vertices indicate actions to execute and the edges indicate transitions conditioned on an observation, or as a finite state controller which executes in a similar manner. An example of each is given in Figure 2.

As in the POMDP case, the goal is to maximize the total cumulative reward, beginning at some initial distribution

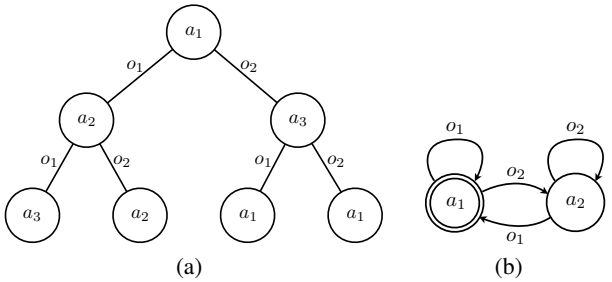


Figure 2: A single agent's policy represented as (a) a policy tree and (b) a finite-state controller with initial state shown with a double circle.

over states b_0. In general, the agents do not observe the actions or observations of the other agents, but the rewards, transitions, and observations depend on the decisions of all agents. The work discussed in this paper (and the vast majority of work in the Dec-POMDP community) considers the case where the model is assumed to be known to all agents.

The value of a joint policy, π, from state s is

$$V^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{h-1} \gamma^t R(\vec{a}^t, s^t) | s, \pi\right],$$

which represents the expected value of the immediate reward for the set of agents summed for each step of the problem given the action prescribed by the policy until the horizon is reached. In the finite-horizon case, the discount factor, γ, is typically set to 1. In the infinite-horizon case, as the number of steps is infinite, the discount factor $\gamma \in [0, 1)$ is included to maintain a finite sum and $h = \infty$. An *optimal policy* beginning at state s is $\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$.

Unfortunately, large problem instances remain intractable: some advances have been made in optimal algorithms [1, 2, 4, 10, 12, 27], but optimally solving a Dec-POMDP is NEXP-complete, so most approaches that scale well make very strong assumptions about the domain (e.g., assuming a large amount of independence between agents) [13, 24, 26] and/or have no guarantees about solution quality [28, 34, 38].

Macro-Actions for Dec-POMDPs

Dec-POMDPs typically require synchronous decision-making: every agent repeatedly determines which action to execute, and then executes it within a single time step. This restriction is problematic for robot domains for two reasons. First, robot systems are typically endowed with a set of controllers, and planning consists of sequencing the execution of those controllers. However, due to both environmental and controller complexity, the controllers will almost always execute for an extended period, and take differing amounts of time to run. Synchronous decision-making would thus require us to wait until all robots have completed their controller execution before we perform the next action selection, which is suboptimal and may not even always be possible (since the robots do not know the system state and staying in place may be difficult in some domains). Second, the planning complexity of a Dec-POMDP is doubly exponential in the horizon. A planner that must try to reason about all of the robots' possible policies at every time step will only ever be able to make very short plans.

Recent research has extended the Dec-POMDP model to plan using *options*, or temporally extended actions [3]. This MacDec-POMDP formulation models a group of robots that must plan by sequencing an existing set of controllers, enabling planning at the appropriate level to compute near-optimal solutions for problems with significantly longer horizons and larger state-spaces.

We can gain additional benefits by exploiting known structure in the multi-robot problem. For instance, most controllers only depend on locally observable information and do not require coordination. For example, consider a controller that navigates a robot to a waypoint. Only local information is required for navigation—the robot may detect other robots but their presence does not change its objective, and it simply moves around them—but choosing the target waypoint likely requires the planner to consider the locations and actions of all robots. Macro-actions with independent execution allow coordination decisions to be made only when necessary (i.e., when choosing macro-actions) rather than at every time step. Because we build on top of Dec-POMDPs, macro-action choice may depend on history, but during execution macro-actions may depend only on a single observation, depend on any number of steps of history, or even represent the actions of a set of robots. That is, macro-actions are very general and can be defined in such a way to take advantage of the knowledge available to the robots during execution.

Model

We first consider macro-actions that only depend on a single robot's information. This is an extension the *options framework* [36] to multi-agent domains while dealing with the lack of synchronization between agents. The options framework is a formal model of a macro-actions [36] that has been very successful in aiding representation and solutions in single robot domains [19]. A MacDec-POMDP with local options is defined as a Dec-POMDP where we also assume M_i represents a finite set of options for each agent, i, with $M = \times_i M_i$ the set of joint options [3]. A *local option* is defined by the tuple:

$$M_i = (\beta_{m_i}, \mathcal{I}_{m_i}, \pi_{m_i}),$$

consisting of stochastic termination condition $\beta_{m_i} : H_i^A \rightarrow [0, 1]$, initiation set $\mathcal{I}_{m_i} \subset H_i^A$ and option policy $\pi_{m_i} : H_i^A \times A_i \rightarrow [0, 1]$. Note that this representation uses action-observation histories of an agent in the terminal and initiation conditions as well as the option policy. Simpler cases can consider reactive policies that map single observations to actions as well as termination and initiation sets that depend only on single observations. This is especially appropriate when the agent has knowledge about aspects of the state necessary for option execution (e.g., its own location when navigating to a waypoint causing observations to be location estimates). As we later discuss, initiation and termi-

nal conditions can depend on global states (e.g., also ending execution based on unobserved events).

Because it may be beneficial for agents to remember their histories when choosing which option to execute, we consider policies that remember option histories (as opposed to action-observation histories). We define an *option history* as

$$H_i^M = (h_i^0, m_i^1, \ldots, h_i^{l-1}, m_i^l),$$

which includes both the action-observation histories where an option was chosen and the selected options themselves. The option history also provides an intuitive representation for using histories within options. It is more natural for option policies and termination conditions to depend on histories that begin when the option is first executed (action-observation histories) while the initiation conditions would depend on the histories of options already taken and their results (option histories). While a history over primitive actions also provides the number of steps that have been executed in the problem (because it includes actions and observations at each step), an option history may require many more steps to execute than the number of options listed. We can also define a (stochastic) local policy, $\mu_i : H_i^M \times M_i \rightarrow [0, 1]$ that depends on option histories. We then define a joint policy for all agents as μ.

Because option policies are built out of primitive actions, we can evaluate policies in a similar way to other Dec-POMDP-based approaches. Given a joint policy, the primitive action at each step is determined by the high level policy which chooses the option and the option policy which chooses the action. The joint policy and option policies can then be evaluated as:

$$V^{\mu}(s) = \mathrm{E}\left[\sum_{t=0}^{h-1} \gamma^t R(\vec{a}^t, s^t) | s, \pi, \mu\right].$$

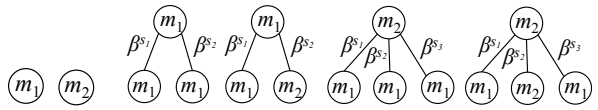
For evaluation in the case where we define a set of options which use observations (rather than histories) for initiation, termination and option policies (while still using option histories to choose options) see Amato, Konidaris and Kaelbling [3].

Algorithms

Because Dec-POMDP algorithms produce policies mapping agent histories to actions, they can be extended to consider options instead of primitive actions. Two such algorithms have been extended [3], but other extensions are possible.

In these approaches, deterministic polices are generated which are represented as policy trees (as shown in Figure 2). A policy tree for each agent defines a policy that can be executed based on local information. The root node defines the option to choose in the known initial state, and another option is assigned to each of the legal terminal states of that option; this continues for the depth of the tree. Such a tree can be evaluated up to a desired (low-level) horizon using the policy evaluation given above, which may not reach some nodes of the tree due to the differing execution times of some options.

A simple exhaustive search method can be used to generate hierarchically optimal deterministic policies. This algorithm is similar in concept to the dynamic programming



(a) Step 1 (b) Step 2 of DP

Figure 3: Policies for a single agent after (a) one step and (b) two steps of dynamic programming using options m_1 and m_2 and (deterministic) terminal states as β^s.

algorithm used in Dec-POMDPs [16], but full evaluation and pruning (removing dominated policies) are not used. Instead the structure of options is exploited to reduce the space of policies considered. That is, to generate deterministic policies, trees are built up as in Figure 3. Trees of increasing depth are constructed until all of the policies are guaranteed to terminate before the desired horizon. When all policies are sufficiently long, all combinations of these policies can be evaluated as above (by flattening out the polices into primitive action Dec-POMDP policies, starting from some initial state and proceeding until h). The combination with the highest value at the initial belief state, b_0, is a hierarchically optimal policy. Note that the benefit of this approach is that only legal policies are generated using the initiation and terminal conditions for options.

Memory-bounded dynamic programming (MBDP) [34] has also been extended to use options as shown in Algorithm 1. This approach bounds the number of trees that are generated by the method above as only a finite number of policy trees are retained (given by parameter *MaxTrees*) at each tree depth. To increase the tree depth to $t + 1$, all possible trees are considered that choose some option and then have the trees retained from depth t as children. Trees are chosen by evaluating them at states that are reachable using a heuristic policy that is executed for the first $h - t - 1$ steps of the problem. A set of $MaxTrees$ states is generated and the highest-valued trees for each state are kept. This process continues, using shorter heuristic policies until all combinations of the retained trees reach horizon h. Again, the set of trees with the highest value at the initial belief state is returned.

The MBDP-based approach is potentially suboptimal because a fixed number of trees are retained, and trees optimized at the states provided by the heuristic policy may be suboptimal (because the heuristic policy may be suboptimal and the algorithm assumes the states generated by the heuristic policy are known initial states for the remaining policy tree). Nevertheless, since the number of policies at each step is bounded by $MaxTrees$, MBDP has time and space complexity linear in the horizon. As a result, this approach has been shown to work well in many large MacDec-POMDPs [3].

Solving Multi-Robot Problems with MacDec-POMDPs

The MacDec-POMDPs framework is a natural way to represent and generate behavior for general multi-robot systems. A high-level description of this process is given in

Algorithm 1 Option-based memory bounded dynamic programming

function OPTIONMBDP($MaxTrees$,h,H_{pol})
  $t \leftarrow 0$
  $someTooShort \leftarrow true$
  $\mu_t \leftarrow \emptyset$
  repeat
    $\mu_{t+1} \leftarrow$GeneateNextStepTrees(μ_t)
    Compute $V^{\mu_{t+1}}$
    $\hat{\mu}_{t+1} \leftarrow \emptyset$
    for all $k \in MaxTrees$ **do**
      $s_k \leftarrow$ GenerateState(H_{pol},$h-t-1$)
      $\hat{\mu}_{t+1} \leftarrow \hat{\mu}_{t+1} \cup \arg\max_{\mu_{t+1}} V^{\mu_{t+1}}(s_k)$
    end for
    $t \leftarrow t+1$
    $\mu_t \leftarrow \hat{\mu}_{t+1}$
    $someTooShort \leftarrow$testLength(μ_t)
  until $someTooShort = false$
  return μ_t
end function

Figure 4. We assume an abstract model of the system is given in the form of macro-action representations, which include the associated policies as well as initiation and terminal conditions. These macro-actions are controllers operating in (possibly) continuous time with continuous actions and feedback, but their operation is discretized for use with the planner. This discretization represents an underlying discrete Dec-POMDP which consists of the primitive actions, states of the system and the associated rewards. The Dec-POMDP methods discussed above typically assume a full model is given, but in this work, we make the more realistic assumption that we can simulate the macro-actions in an environment that is similar to the real-world domain. As a result, we do not need a full representation of the underlying Dec-POMDP and use the simulator to test macro-action completion and evaluate policies. In the future, we plan to remove this underlying Dec-POMDP modeling and instead represent the macro-action initiation, termination and policies using features directly in the continuous robot state-space. In practice, models of each macro-action's behavior can be generated by executing the corresponding controller from a variety of initial conditions (which is how our model and simulator was constructed in the experiment section). Given the macro-actions and simulator, the planner then automatically generates a solution which optimizes the value function with respect to the uncertainty over outcomes, sensor information and other agents. This solution comes in the form of SMACH controllers [9] which are hierarchical state machines for use in a ROS [32] environment. Each node in the SMACH controller represents a macro-action which is executed on the robot and each edge corresponds to a terminal condition. In this way, the trees in Figure 3 can be directly translated into SMACH controllers, one for each robot. Our system is thus able to automatically generate SMACH controllers, which are typically designed by hand, for complex, general multi-robot systems.

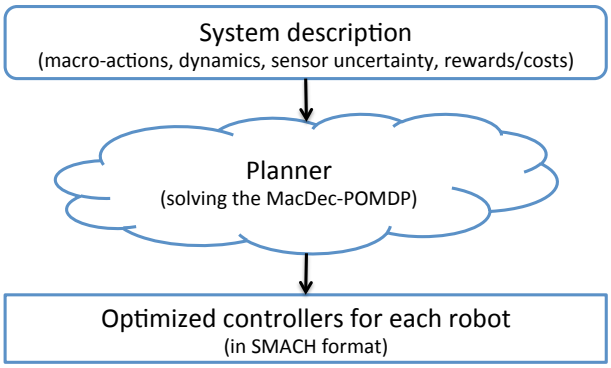


Figure 4: A high level system diagram.

It is also worth noting that our approach can incorporate existing solutions for more restricted scenarios as macro-actions. For example, our approach can build on the large amount of research in single and multi-robot systems that has gone into solving difficult problems such as navigation in a formation [5] or cooperative transport of an object [20]. The solutions to these problems could be represented as macro-actions in our framework, building on existing research to solve even more complex multi-robot problems.

Planning using MacDec-POMDPs in the Warehouse Domain

We test our methods in a warehousing scenario using a set of iRobot Creates (Figure 5), and demonstrate how the same general model and solution methods can be applied in versions of this domain with different communication capabilities. This is the first time that Dec-POMDP-based methods have been used to solve large multi-robot domains. We do not compare with other methods because other Dec-POMDP cannot solve problems of this size and current multi-robot methods cannot automatically derive solutions for these multifaceted problems. The results demonstrate that our methods can automatically generate the appropriate motion and communication behavior while considering uncertainty over outcomes, sensor information and other robots.

The Warehouse Domain

We consider three robots in a warehouse that are tasked with finding and retrieving boxes of two different sizes: large and small. Robots can navigate to known depot locations (rooms) to retrieve boxes and bring them back to a designated drop-off area. The larger boxes can only be moved effectively by two robots (if a robot tries to pick up the large box by itself, it will move to the box, but fail to pick it up). While the locations of the depots are known, the contents (the number and type of boxes) are unknown. Our planner generates a SMACH controller for each of the robots offline which are then executed online in a decentralized manner.

In each scenario, we assumed that each robot could observe its own location, see other robots if they were within (approximately) one meter, observe the nearest box when

Figure 5: The warehouse domain with three robots.

in a depot and observe the size of the box if it is holding one. These observations were implemented within a Vicon setup to allow for system flexibility, but the solutions would work in any setting in which these observations are generated. In the simulator that is used by the planner to generate and evaluate solutions, the resulting state space of the problem includes the location of each robot (discretized into nine possible locations) and the location of each of the boxes (in a particular depot, with a particular robot or at the goal). The primitive actions are to move in four different directions as well as pickup, drop and communication actions. Note that this primitive state and action representation is used for evaluation purposes and not actually implemented on the robots (which just utilize the SMACH controllers). Higher fidelity simulators could also be used. The three-robot version of this scenario has 1,259,712,000 states, which is several orders of magnitude larger than problems typically solvable by Dec-POMDP solvers. These problems are solved using the option-based MBDP algorithm initialized with a hand coded heuristic policy. Experiments were run on a single core of a 2.5 GHz machine with 8GB of memory. Exact solution times were not calculated, but average solution times for the policies presented below were approximately one hour.

In our Dec-POMDP model, navigation has a small amount of noise in the amount of time required to move to locations (reflecting the real-world dynamics): this noise increases when the robots are pushing the large box (reflecting the need for slower movements and turns in this case). We defined macro-actions that depend only on the observations above, but option selection depends on the history of options executed and observations seen as a result (the option history).

Scenario 1: No Communication

In the first scenario, we consider the case where robots could not communicate with each other. Therefore, all cooperation is based on the controllers that are generated by the planner (which knows the controllers generated for all robots when planning offline) and observations of the other robots (when executing online). The macro-actions were as follows:

- Go to depot 1.
- Go to depot 2.

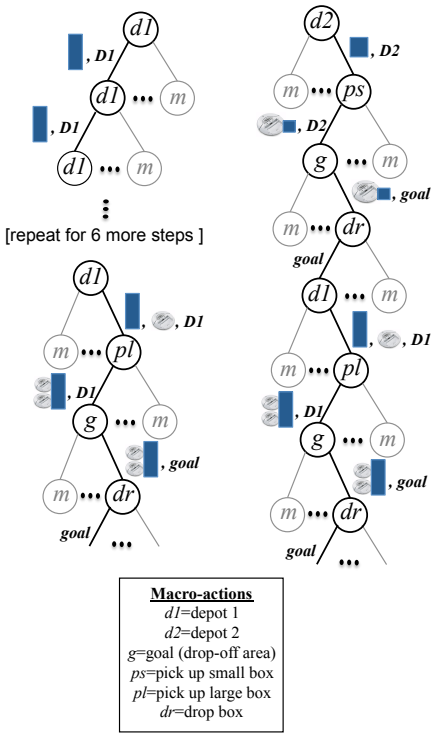


Figure 7: Path executed in policy trees. Only macro-actions executed (nodes) and observations seen (edges, with boxes and robots given pictorially) are shown.

- Go to the drop-off area.
- Pick up the small box.
- Pick up the large box.
- Drop off a box.

The depot macro-actions are applicable anywhere and terminate when the robot is within the walls of the appropriate depot. The drop-off and drop macro-actions are only applicable if the robot is holding a box, and the pickup macro-actions are only applicable when the robot observes a box of the particular type. Navigation is stochastic in the amount of time that will be required to succeed (as mentioned above). Picking up the small box was assumed to succeed deterministically, but this easily be changed if the pickup mechanism is less robust. These macro-actions correspond to natural choices for robot controllers.

This case[1] (seen in Figure 6 along with a depiction of the executed policy in Figure 7) uses only two robots to more clearly show the optimized behavior in the absence of communication. The policy generated by the planner assigns one robot to go to each of the depots (Figure 6(a)). The robots then observe the contents of the depots they are in (Figure 6(b)). If there are two robots in the same room as a large box, they will push it back to the goal. If there is only one robot in a depot and there is a small box to push, the robot

[1]Videos for all scenarios can be seen at `http://youtu.be/istb8TIp_jw`

(a) Two robots set out for different depots.

(b) The robots observe the boxes in their depots (large on left, small on right).

(c) White robot moves to the large box and green robot moves to the small one.

(d) White robot waits while green robot pushes the small box.

(e) Green robot drops the box off at the goal.

(f) Green robot goes to the depot 1 and sees the other robot and large box.

(g) Green robot moves to help the white robot.

(h) The two robots push the large box back to the goal.

Figure 6: Video captures from the no communication version of the warehouse problem.

will push the small box (Figure 6(c)). If the robot is in a depot with a large box and no other robots, it will stay in the depot, waiting for another robot to come and help push the box (Figure 6(d)). In this case, once the the other robot is finished pushing the small box (Figure 6(e)), it goes back to the depots to check for other boxes or robots that need help (Figure 6(f)). When it sees another robot and the large box in the depot on the left (depot 1), it attempts to help push the large box (Figure 6(g)) and the two robots are successful pushing the large box to the goal (Figure 6(h)). In this case, the planner has generated a policy in a similar fashion to task allocation—two robots go to each room, and then search for help needed after pushing any available boxes. However, in our case this behavior was generated by an optimization process that considered the different costs of actions, the uncertainty involved and the results of those actions into the future.

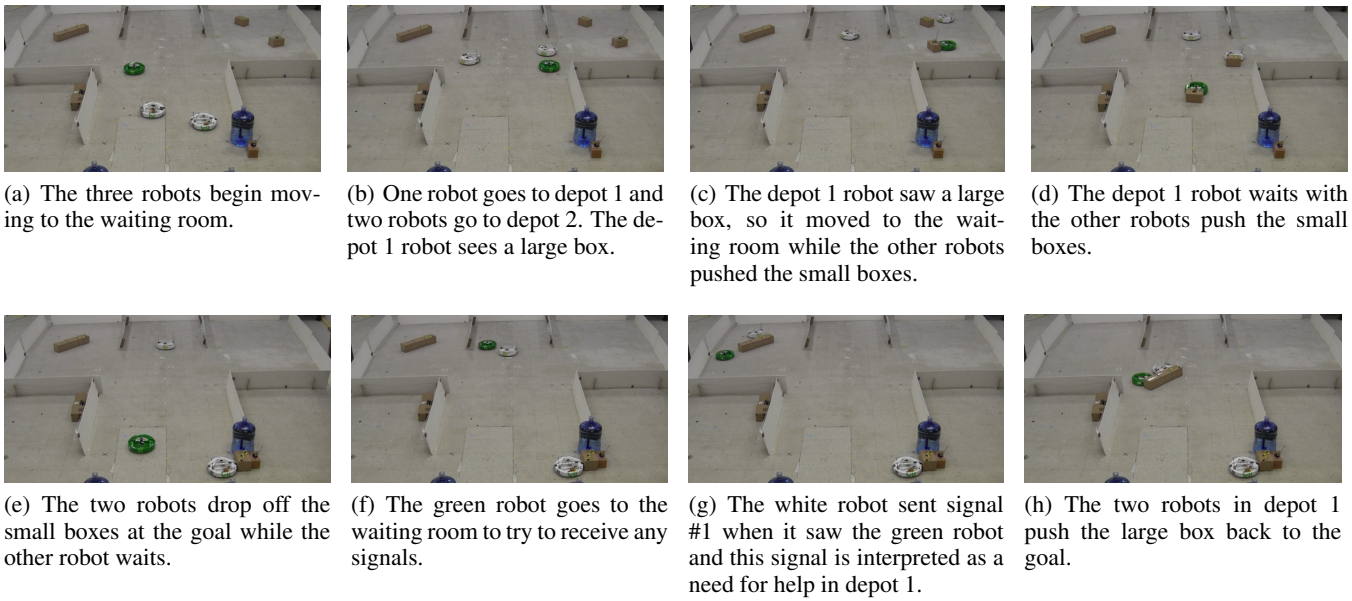
Scenario 2: Local Communication

In scenario 2, robots can communicate when they are within one meter of each other. The macro-actions are the same as above, but we added ones to communicate and wait for communication. The resulting macro-action set is:

- Go to depot 1.
- Go to depot 2.
- Go to the drop-off area.
- Pick up the small box.
- Pick up the large box.
- Drop off a box.
- Go to an area between the depots (the "waiting room").
- Wait in the waiting room for another robot.
- Send signal #1.
- Send signal #2.

Here, we allow the robots to choose to go to a "waiting room" which is between the two depots. This permits the robots to possibly communicate or receive communications before committing to one of the depots. The waiting-room macro-action is applicable in any situation and terminates when the robot is between the waiting room walls. The depot macro-actions are now only applicable in the waiting room, while the drop-off, pick up and drop macro-actions remain the same. The wait macro-action is applicable in the waiting room and terminates when the robot observes another robot in the waiting room. The signaling macro-actions are applicable in the waiting room and are observable by other robots that are within approximately a meter of the signaling robot. Note that *we do not specify what sending each communication signal means*.

The results for this domain are shown in Figure 8. We see that the robots go to the waiting room (Figure 8(a)) (because we required the robots to be in the waiting room before choosing to move to a depot) and then two of the robots go to depot 2 (the one on the right) and one robot goes to depot 1 (the one on the left) (Figure 8(b)). Note that because there are three robots, the choice for the third robot is random while one robot will always be assigned to each of the depots. Because there is only a large box to push in depot 1, the robot in this depot goes back to the waiting room to try to find another robot to help it push the box (Figure 8(c)). The robots in depot 2 see two small boxes and they choose to push these back to the goal (Figure 8(d)). Once the small boxes are dropped off (Figure 8(e)), one of the robots returns to the waiting room (Figure 8(f)) and then is recruited by the other robot to push the large box back to the goal (Figure 8(g)). The robots then successfully push the large box back to the goal (Figure 8(h)). Note that in this case the planning process *determines how the signals should be used to*



(a) The three robots begin moving to the waiting room.

(b) One robot goes to depot 1 and two robots go to depot 2. The depot 1 robot sees a large box.

(c) The depot 1 robot saw a large box, so it moved to the waiting room while the other robots pushed the small boxes.

(d) The depot 1 robot waits with the other robots push the small boxes.

(e) The two robots drop off the small boxes at the goal while the other robot waits.

(f) The green robot goes to the waiting room to try to receive any signals.

(g) The white robot sent signal #1 when it saw the green robot and this signal is interpreted as a need for help in depot 1.

(h) The two robots in depot 1 push the large box back to the goal.

Figure 8: Video captures from the limited communication version of the warehouse problem.

perform communication.

Scenario 3: Global Communication

In the last scenario, the robots can use signaling (rather than direct communication). In this case, there is a switch in each of the depots that can turn on a blue or red light. This light can be seen in the waiting room and there is another light switch in the waiting room that can turn off the light. (The light and switch were simulated in software and not incorporated in the physical domain.) As a result, the macro-actions in this scenario were as follows:

- Go to depot 1.
- Go to depot 2.
- Go to the drop-off area.
- Pick up the small box.
- Pick up the large box.
- Drop off a box.
- Go to an area between the depots (the "waiting room").
- Turn on a blue light.
- Turn on a red light.
- Turn off the light.

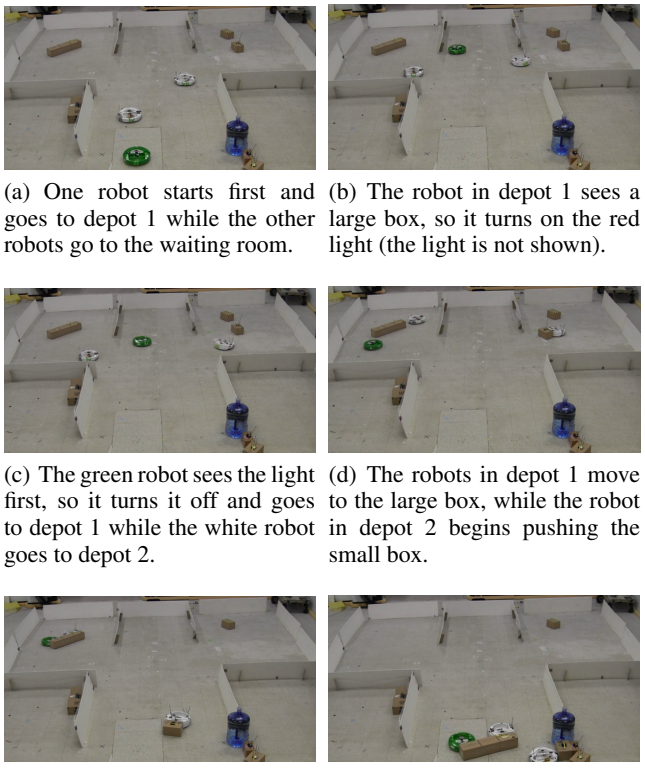
The first seven macro-actions are the same as for the communication case except we relaxed the assumption that the robots had to go to the waiting room before going to the depots (making both the depot and waiting room macro-actions applicable anywhere). The macro-actions for turning the lights on are applicable in the depots and the macro-actions for turning the lights off are applicable in the waiting room. While the lights were intended to signal requests for help in each of the depots, we did not assign a particular color to a particular depot. In fact, we did not assign them any specific meaning, allowing the planner to set them in any way that improves performance.

The results are shown in Figure 9. Because one robot started ahead of the others, it was able to go to depot 1 to sense the size of the boxes while the other robots go to the waiting room (Figure 9(a)). The robot in depot 1 turned on the light (red in this case, but not shown in the images) to signify that there is a large box and assistance is needed (Figure 9(b)). The green robot (the first other robot to the waiting room) sees this light, interprets it as a need for help in depot 1, and turns off the light (Figure 9(c)). The other robot arrives in the waiting room, does not observe a light on and moves to depot 2 (also Figure 9(c)). The robot in depot 2 chooses to push a small box back to the goal and the green robot moves to depot 1 to help the other robot (Figure 9(d)). One robot then pushes the small box back to the goal while the two robots in depot 1 begin pushing the large box (Figure 9(e)). Finally, the two robots in depot 1 push the large box back to the goal (Figure 9(f)). This behavior is optimized based on the information given to the planner. *The semantics of all these signals as well as the movement and signaling decisions were decided on by the planning algorithm to maximize value.*

Related Work

There are several frameworks that have been developed for multi-robot decision making in complex domains. For instance, behavioral methods have been studied for performing task allocation over time in loosely-coupled [30] or tightly-coupled [35] tasks. These are heuristic in nature and make strong assumptions about the type of tasks that will be completed.

One important related class of methods is based on using



(a) One robot starts first and goes to depot 1 while the other robots go to the waiting room.

(b) The robot in depot 1 sees a large box, so it turns on the red light (the light is not shown).

(c) The green robot sees the light first, so it turns it off and goes to depot 1 while the white robot goes to depot 2.

(d) The robots in depot 1 move to the large box, while the robot in depot 2 begins pushing the small box.

(e) The robots in depot 1 begin pushing the large box and the robot in depot 2 pushes a small box to the goal.

(f) The robots from depot 1 successfully push the large box to the goal.

Figure 9: Video captures from the signaling version of the warehouse problem.

linear temporal logic (LTL) [7, 21] to specify behavior for a robot; from this specification, reactive controllers that are guaranteed to satisfy the specification can be derived. These methods are appropriate when the world dynamics can be effectively described non-probabilistically and when there is a useful discrete characterization of the robot's desired behavior in terms of a set of discrete constraints. When applied to multiple robots, it is necessary to give each robot its own behavior specification. Other logic-based representations for multi-robot systems have similar drawbacks and typically assume centralized planning and control [22].

Market-based approaches use traded value to establish an optimization framework for task allocation [11, 15]. These approaches have been used to solve real multi-robot problems [18], but are largely aimed to tightly-coupled tasks, where the robots can communicate through a bidding mechanism.

Emery-Montemerlo et al. [14] introduced a (cooperative) game-theoretic formalization of multi-robot systems which resulted in solving a Dec-POMDP. An approximate forward search algorithm was used to generate solutions, but scalability was limited because a (relatively) low-level Dec-POMDP was used. Also, Messias et al. [25] introduce an MDP-based model where a set of robots with controllers that can execute for varying amount of time must cooperate to solve a problem. However, decision-making in their system is centralized.

Conclusion

We have demonstrated—for the first time—that complex multi-robot domains can be solved with Dec-POMDP-based methods. The MacDec-POMDP model is expressive enough to capture multi-robot systems of interest, but also simple enough to be feasible to solve in practice. Our results show that a general purpose MacDec-POMDP planner can generate cooperative behavior for complex multi-robot domains with task allocation, direct communication, and signaling behavior emerging automatically as properties of the solution for the given problem model. Because all cooperative multi-robot problems can be modeled as Dec-POMDPs, MacDec-POMDPs represent a powerful tool for automatically trading-off various costs, such as time, resource usage and communication while considering uncertainty in the dynamics, sensors and other robot information. These approaches have great potential to lead to automated solution methods for general multi-robot coordination problems with large numbers of heterogeneous robots in complex, uncertain domains.

In the future, we plan to explore incorporating these state-of-the-art macro-actions into our MacDec-POMDP framework as well as examine other types of structure that can be exploited. Other topics we plan to explore include increasing scalability by making solution complexity depend on the number of agent interactions rather than the domain size, and having robots learn models of their sensors, dynamics and other robots. These approaches have great potential to lead to automated solution methods for general multi-robot coordination problems with large numbers of heterogeneous robots in complex, uncertain domains.

References

[1] Amato, C.; Chowdhary, G.; Geramifard, A.; Ure, N. K.; and Kochenderfer, M. J. 2013. Decentralized control of partially observable Markov decision processes. In *Proceedings of the Fifty-Second IEEE Conference on Decision and Control*, 2398–2405.

[2] Amato, C.; Dibangoye, J. S.; and Zilberstein, S. 2009. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, 2–9.

[3] Amato, C.; Konidaris, G.; and Kaelbling, L. P. 2014. Planning with macro-actions in decentralized POMDPs. In *Proceedings of the Thirteenth International Conference on Autonomous Agents and Multiagent Systems*.

[4] Aras, R.; Dutech, A.; and Charpillet, F. 2007. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, 18–25.

[5] Balch, T., and Arkin, R. C. 1998. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation* 14(6):926–939.

[6] Beckers, R.; Holland, O.; and Deneubourg, J.-L. 1994. From local actions to global tasks: Stigmergy and collective robotics. In *Artificial life IV*, volume 181, 189.

[7] Belta, C.; Bicchi, A.; Egerstedt, M.; Frazzoli, E.; Klavins, E.; and Pappas, G. J. 2007. Symbolic planning and control of robot motion [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE* 14(1):61–70.

[8] Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27(4):819–840.

[9] Bohren, J. 2010. SMACH. `http://wiki.ros.org/smach/`.

[10] Boularias, A., and Chaib-draa, B. 2008. Exact dynamic programming for decentralized POMDPs with lossless policy compression. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*.

[11] Dias, M. B., and Stentz, A. T. 2003. A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, volume 3, 2279 – 2284.

[12] Dibangoye, J. S.; Amato, C.; Buffet, O.; and Charpillet, F. 2013a. Optimally solving Dec-POMDPs as continuous-state MDPs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[13] Dibangoye, J. S.; Amato, C.; Doniec, A.; and Charpillet, F. 2013b. Producing efficient error-bounded solutions for transition independent decentralized MDPs. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems*.

[14] Emery-Montemerlo, R.; Gordon, G.; Schneider, J.; and Thrun, S. 2005. Game theoretic control for robot teams. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 1163–1169.

[15] Gerkey, B., and Matarić, M. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research* 23(9):939–954.

[16] Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 709–715.

[17] Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:1–45.

[18] Kalra, N.; Ferguson, D.; and Stentz, A. T. 2005. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proceedings of the International Conference on Robotics and Automation*, 1170 – 1177.

[19] Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11):1238 – 1274.

[20] Kube, C., and Bonabeau, E. 2000. Cooperative transport by ants and robots. *Robotics and Autonomous Systems* 30(1-2):85–101.

[21] Loizou, S. G., and Kyriakopoulos, K. J. 2004. Automatic synthesis of multi-agent motion tasks based on ltl specifications. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, 153–158. IEEE.

[22] Lundh, R.; Karlsson, L.; and Saffiotti, A. 2008. Autonomous functional configuration of a network robot system. *Robotics and Autonomous Systems* 56(10):819–830.

[23] Mahajan, A. 2013. Optimal decentralized control of coupled subsystems with control sharing. *IEEE Transactions on Automatic Control* 58:2377–2382.

[24] Melo, F. S., and Veloso, M. 2011. Decentralized MDPs with sparse interactions. *Artificial Intelligence*.

[25] Messias, J. V.; Spaan, M. T.; and Lima, P. U. 2013. GSMDPs for multi-robot sequential decision-making. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

[26] Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*.

[27] Oliehoek, F. A.; Spaan, M. T. J.; Amato, C.; and Whiteson, S. 2013. Incremental clustering and expansion for faster optimal planning in Dec-POMDPs. *Journal of Artificial Intelligence Research* 46:449–509.

[28] Oliehoek, F. A.; Whiteson, S.; and Spaan, M. T. J. 2013. Approximate solutions for factored Dec-POMDPs with many agents. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems*.

[29] Oliehoek, F. A. 2012. Decentralized POMDPs. In Wiering, M., and van Otterlo, M., eds., *Reinforcement Learning: State of the Art*, volume 12 of *Adaptation, Learning, and Optimization*. Springer. 471–503.

[30] Parker, L. E. 1998. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation* 14(2):220–240.

[31] Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.

[32] Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an

open-source robot operating system. In *ICRA workshop on open source software*, volume 3.

[33] Rekleitis, I.; Lee-Shue, V.; New, A. P.; and Choset, H. 2004. Limited communication, multi-robot team based coverage. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, 3462–3468. IEEE.

[34] Seuken, S., and Zilberstein, S. 2007. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2009–2015.

[35] Stroupe, A. W.; Ravichandran, R.; and Balch, T. 2004. Value-based action selection for exploration and dynamic target observation with robot teams. In *Proceedings of the International Conference on Robotics and Automation*, volume 4, 4190–4197. IEEE.

[36] Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1):181–211.

[37] Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[38] Velagapudi, P.; Varakantham, P. R.; Sycara, K.; and Scerri, P. 2011. Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*.

Selecting Paths to Minimize Conflicts in Crowded Scenes using Minimal Information

Andrew Kimmel and **Kostas Bekris**
Department of Computer Science, Rutgers University
Piscataway, New Jersey 08854

Abstract

Consider multiple robots moving towards individual goals in a cluttered environment. While contacts between robots in these situations can be averted by reactive collision avoidance methods, deadlocks may arise in tight spaces if robots move along precomputed, conflicting paths. To resolve these issues, methods have been proposed which consider robots that employ communication, or centralized planning, or follow predefined rules. This work considers only decentralized planning solutions that employ minimum information, i.e., each robot has access only to the current position of its neighbors, without using any form of prediction, intent recognition or agent modeling. This leads to a study of several methods for minimum-conflict path selection among dynamic obstacles. The evaluation of these methods in varying simulated benchmarks, provides the following insights: (a) considering the minimum-conflict path given the other agents current positions is critical for deadlock avoidance, (b) reasoning over a diverse set of paths that cover the workspace improves path quality, and (c) accumulating cost over these paths by finding the agent's true optimal path in hindsight allows robots to optimize and learn effective high-level strategies in a computationally efficient way that is adaptive to the other agents behavior, reducing task completion time and average path lengths.

Introduction

The proliferation of robotic technology allows for consideration of applications where multiple autonomous systems effortlessly interact in the same cluttered environment, while solving individual tasks. For example, consider ground vehicles that operate in a mine or a construction facility and which need to move efficiently to solve their assigned task while avoiding collisions. In many interesting challenges, it is also highly likely that people or animals are navigating in the same space and need to be considered. Explicit coordination with the other agents in the environment, especially when humans or animals are involved, may not feasible nor desirable. Similarly, it may be difficult to model or predict the actions of moving obstacles. This necessitates the consideration of decentralized methods that allow a robot to make progress towards its goal in a safe manner with minimal information and without strong assumptions about the intentions of its moving neighbors. It is also desirable for the solution to minimize certain parameters, such as executed path length or task completion time, and for the motions to look natural to people that operate in the same space.

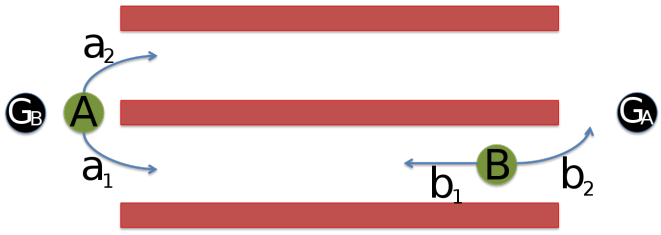


Figure 1: The base case for this study: if both robots A and B insist on following the same corridor, then given the environment characteristics and their geometry, a reactive collision avoidance method may not allow them to make progress towards their goals, G_A and G_B, which are on opposite sides.

In environments without obstacles, reactive methods are able to perform well even for large numbers of agents. However, if the environment also contains a complex set of obstacles, it becomes necessary to utilize a planner to compute the solution path. Solving problems that combines these properties, complex environments with dense distributions of agents, thus necessitates the use of both a high-level planner, operating in a replanning framework with a reasonably small planning cycle (so as to adapt to the frequent changes in such scenarios), as well as a low-level reactive collision avoidance technique to account for the planning cycle time. This work was originally submitted to IROS 2014.

Challenges, Foundations and Objectives

Avoiding collisions with unexpected obstacles or unpredictable, self-interested mobile agents, can be effectively addressed by reactive collision avoidance methods, such as those based on the popular Velocity Obstacle framework [Fiorini and Shiller1998, van den Berg et al.2011] or trajectory deformation methods [Fraichard and Delsart2009, Karamouzas, Geraerts, and Overmars2009]. These methods generally provide smooth, natural-looking paths, but they are primarily local techniques and do not reason about the robot's global path. If the agents select conflicting paths in a decentralized manner, reactive collision avoidance can still

give rise to deadlocks and poor performance. For instance, consider the situation in Figure 1, where two robots on opposing sides of two corridors need to exchange positions. If both robots decide to move along the lower corridor, e.g., because it corresponds to their individual shortest paths, the space is narrow enough to prevent the robots from swapping positions.

Robots in such situations that replan [Petti and Fraichard2005] and change their path to a different homotopic class [Bhattacharya, Kumar, and Likhachev2010, Bhattacharya, Likhachev, and Kumar2011, Jaillet and Siméon2006] can potentially resolve such conflicts. This type of coordination can be achieved by assuming that the robots share information [Bekris et al.2012], or follow a form of centralized planning [Qutub, Alami, and Ingrand1997], or by respecting a set of pre-specified "social" rules [Knepper and Rus2012, Trautman and Krause2010], or performing sophisticated prediction [Large et al.2004, Thompson, Horiuchi, and Kagami2009, Ziebart et al.2009], agent modeling [Shi et al.2008, Sisbot et al.2007] or learning [Bennewitz et al.2005, Henry et al.2010]. The type of information required in order to be able to use such solutions may correspond a)to the actions selected by neighbors, b) the utilities of different motions, c)the goals of neighbors, or d) extensive prior experience interacting with other agents. Such information is difficult to attain quickly and reliably, especially when a robot interacts with a human, since the robot has little knowledge about the human's future actions without explicit communication. This work employs strictly decentralized methods while utilizing minimal information to solve this problem. Each robot has access only to the current position of its neighbors from sensor data. A further objective is to identify what can be achieved without any prediction, intent recognition or modeling of the moving agents. This method employs learning, corresponding primarily to online learning of appropriate strategies in response to environmental conditions.

This planning-based work is complementary to reactive collision avoidance methods and shares the desire of requiring minimal information and assumptions about the other agents. The Velocity Obstacles framework requires knowledge of the velocity of neighboring agents, which is not utilized by the methods proposed here. Reciprocity between the agents utilizing the same method is desirable, as illustrated with Reciprocal Velocity Obstacles [Snape et al.2011, van den Berg et al.2011], so that in situations as the one depicted in Figure 1, one of the agents will reliably decide to switch corridor even if both agents originally selected to move along the same one.

Considered Methodology

The above objectives lead to the consideration of various methods for computing decentralized, minimum-information, minimum-conflict path selection strategies. These methods are evaluated in terms of their effectiveness in various simulated benchmarks. The basic framework assumes that the robots follow a replanning approach to compute paths frequently [Petti and Fraichard2005] and then they employ reciprocal velocity obstacles [van den Berg, Lin, and Manocha2008, Snape et al.2011] so as to follow these paths while avoiding collisions. If agents make greedy choices to use the shortest path to their goal and ignore the presence of other agents, this approach leads to deadlocks in the considered environments.

One idea is to compute a family of diverse paths instead of only the shortest one. The notion of path diversity has been shown to be helpful in different challenges, but frequently corresponds to a local concept [Green and Kelly2007, Knepper and Mason2009]. If a roadmap for the scene is available, then an effective way to compute this set of paths is to compute trajectories belonging to different homotopic classes, using search-based primitives [Bhattacharya, Kumar, and Likhachev2010, Bhattacharya, Likhachev, and Kumar2011]. This work provides a method for selecting a minimally conflicting path out of this set given the other agents' current positions. This process is designed so as to effectively and reliably address the issue in the prototypical example provided in Figure 1. If the replanning cycle of the robot is short, this process yields a high-level of reactivity and deconfliction.

While this is an improvement over considering only the shortest path, it also causes computational issues in scenes with many homotopic classes. If only a small set of short in length, homotopically-distinct paths are generated each planning cycle, deadlocks still arise. For instance, this occurs if none of the paths permit the robot to backtrack so as to allow other agents to make progress. Nevertheless, there is a way to address this issue according to simulated results. If the "minimum-conflict" homotopy is always included in the set of considered actions, deadlocks are not observed. The notion of minimum conflict considered here is related to the "minimum constraint displacement" problem, which has attracted attention recently in motion planning [Hauser2013]. For the current work, constraints on the minimum-conflict homotopy correspond to the observed locations of other agents. This homotopy can be discovered in a computationally effective manner. Using an underlying roadmap, the method computes the shortest path that also minimizes the number of collisions with other agents.

While minimum conflict paths achieve deadlock avoidance, they are also conservative, and may take a long time to reach the robots' goals. Considering both the minimum-conflict path and a set of shortest, homotopically-distinct paths towards the goal that ignore the other agents typically results in better performance. Under the "garage" assumption, where agents essentially "disappear" from the workspace when they reach their goals, and assuming that the workspace is unbounded, by strictly navigating using only minimum conflict paths, results seem to indicate that liveness can be guaranteed. This work considers both a deterministic approach for choosing between these paths, as well as a learning-based, probabilistic approach. Both solutions always find a solution and avoid deadlocks in simulations. The probabilistic approach is also able to provide adaptivity to various behaviors of neighbors.

The probabilistic solution is based on the Polynomial Weights `PW` algorithm [Littlestone and Warmuth1994, Nisan et al.2007], which is a learning-based approach for achieving regret minimization. Given an individual agent's action

set, PW accumulates a weight on each action which directly corresponds to the action's probability of being selected. At each step of the framework, an agent evaluates its most recently executed action and computes the regret for the action, which is evaluated by considering the current state of the world compared to the previous state. Essentially, the agent thinks "What would have happened if I selected a different action, given the current observed positions of the other agents?", and in doing so, the agent can compute compute its best action in hindsight. The difference in action cost between this best action in hindsight and the actual selected action, is represented by the agent's regret value, which will directly reduce the action's weight and thus its probability of selection.

Regret minimization is advantageous in this context, as it results in high expected utilities against unpredictable agents, without knowledge of the other agents' goals, utilities, intents, or beliefs. The application of the PW algorithm here accumulates regret for the "minimum-conflict" strategy and the "greedy" choice strategy at each replanning cycle by observing the choices of the other agents in the same workspace and assigning a loss to each strategy. A probability is then assigned for selecting each strategy based on the accumulated losses.

Such learning-based approaches to coordination and game theoretic challenges have been considered by others in the robotics literature, where a reinforcement learning method has been used to create adaptive, loosely coupled, agents [Kaminka, Erusalimchik, and Kraus2010]. Some approaches qualitatively measure the effectiveness of coordination between agents offline to provide action selection online [Excelente-Toledo and Jennings2004]. There has been work in coordinating with agents that are not necessarily rational [Stone, Kaminka, and Rosenschein2010]. All of these approaches require knowledge of the agents' actions, which is not assumed here. Although there are models for predicting the actions of an agent, such as in an adversarial setting [Wunder et al.2011], by utilizing regret minimization [Filiot, Le Gall, and Raskin2010], it is possible to solve certain games without knowing the other agent's actions.

Contribution and Overview of Results

The key observations from this work are the following:

a) Computing minimum-conflict paths is critical for avoiding deadlocks under a minimum information setup for planning among dynamic obstacles and is interesting to further analyze the properties of this strategy.

b) Computing paths in different homotopies is a useful primitive for providing diverse alternatives to robots to improve the resulting path quality and execution time.

c) Regret minimization is computationally efficient and allows robots to optimize and learn over time an appropriate strategy given the characteristics of the underlying challenge, without explicit communication.

Simulations show that the considered solutions allow robots to avoid deadlocks, minimize completion time and path length for solving such problems with minimal information requirements.

Problem Setup

Path deconfliction problems can be defined in general configuration spaces but this discussion will focus on holonomic navigation as it provides an easy way to describe the framework and corresponds to the accompanying simulations.

Consider a set of m planar, holonomic agents $\{\mathcal{A}_1, \ldots, \mathcal{A}_m\}$ that move with bounded velocity $v \in [0, v_{max}]$ in the same workspace $\mathcal{W}$. The configuration space of an agent is $\mathcal{Q} = \mathbb{R}^2$, where $\mathcal{Q}_{free}$ represents the obstacle free subset given static obstacles. Given a configuration $q_i \in \mathcal{Q}$, the expression $\mathcal{A}(q_i)$ corresponds to the collision volume of agent $\mathcal{A}_i$ in $\mathcal{W}$.

Then a path $\pi_i = \{q_i | q_i : [0,1] \rightarrow Q_{free}\}$ for agent $\mathcal{A}_i$ corresponds to a continuous curve in Q_{free}. Given a time scaling function $\sigma_i : \mathbb{R}^{\geq 0} \rightarrow [0,1]$ it is also possible to define the agent's trajectory $\tau_i = \pi_i \circ \sigma_i$, which defines the configurations that the agent visits at each point in time.

The problem formulation assumes that all agents want to solve a similar problem, as in each agent $\mathcal{A}_i$ wants to reach a desired goal $q_i^{\mathcal{G}} \in \mathcal{Q}$ without conflicts. The objective then is for the agents to select trajectories $\{\tau_1, \ldots, \tau_m\}$ in a decentralized manner, such that each $\mathcal{A}_i$ an individual τ_i and in finite time T: $\forall\, i \in [1,m] : \tau_i[T] = q_i^{\mathcal{G}}$. Collisions between agents must be avoided, unless one of the agents has reached its goal, i.e. $\forall\, t \in \mathbb{R}^{\geq 0}, \forall\, i,j \in [1,m]$:

$$\mathcal{A}(\tau_i[t]) \cap \mathcal{A}(\tau_j[t]) = \emptyset \ \vee \ \mathcal{A}(\tau_i[t]) = q_i^{\mathcal{G}} \ \vee \ \mathcal{A}(\tau_j[t]) = q_j^{\mathcal{G}}.$$

The above expression implies the so called "garage" assumption, where agents which reach their goal are freed from the workspace and are not considered for collisions when other agents pass through their goal.

Agents are never aware of the goal of any other agent or the trajectory selected by another agent. At any point in time an agent can only observe the position of other agents as long as their configurations are within a certain sensing radius: $||(q_i, q_j)|| \leq d^{sense}$.

Furthermore, the agents are assumed to be equipped with a collision avoidance method (e.g., Reciprocal Velocity Obstacles [van den Berg, Lin, and Manocha2008]), which is used to follow their selected trajectory while still avoiding collisions with other agents. This means that the planned trajectory may not be executed perfectly due to the influence of neighboring agents.

Note that the above discussion can be easily extended to include the case where one agent is the planning robot that employs a method for achieving deconfliction while all the other agents are unpredictable dynamic obstacles that ignore the presence of the planning agent. In these situations, the relative velocity of the planning agent and the dynamic obstacles should be such so that the collision avoidance method can always guarantee the safety of the planning agent.

The above fact, together with the need to adapt to the unpredictable behavior of neighboring agents, motivates a replanning framework for recomputing trajectories given the latest observed configurations of agents. This replanning approach forms the basis of the overall methodology that is described in the following section.

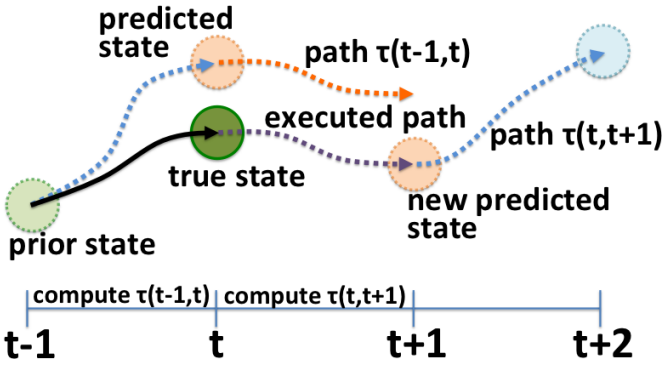


Figure 2: An illustration of the replanning framework. Here the path computed between time t-1 and t is executed during time t to t+1. Execution of the plan means the state at time t deviates from the predicted state, so the framework begins planning for t+1 to t+2 from an updated predicted state.

Methods

This section first describes a classical method for integrating global path planning and local collision avoidance, which can lead to deadlocks in certain environments. Then a sequence of alternative strategies for computing the global path are considered so as to avoid such situations.

Replanning Framework

During the execution of a plan, a robot's trajectory will deviate from the planned trajectory due to the reactive collision avoidance utilized. Naïvely following the original planned trajectory is therefore not sufficient, as the robot will most likely not be able to reach its goal as intended. A straightforward replanning framework [Petti and Fraichard2005,Hauser2011] as illustrated in Figure 2 is used to address such issues. The framework follows related work, where first, a roadmap is precomputed using a sampling-based motion planning method and then integrated with a collision-avoidance method [van Den Berg et al.2008]. The sampling-based planner used in this work is PRM* [Karaman and Frazzoli2011].

The trajectory computed for time t-1 to t will not be executed perfectly, as shown in Figure 2; however, the framework updates the predicted state of the robot accordingly. From the agent's state at time t-1, the robot created a plan which should have brought it to the predicted state at time t, but the collision avoidance led it instead to its true state. The plan for time t to t+1 is then propagated from the true state of the robot to obtain a new predicted state, and this state for time t+1 is used to plan for time t+1 to t+2. In this way, the robot can iteratively correct deviations from its plan, applying this continuously until it is able to reach its goal.

By using such a replanning framework, where the agent assumes its previous selected plan will be executed perfectly and plans for the next time step, the agent becomes robust to perturbations in the solution path caused by these reactive methods. When integrated with a reactive collision avoidance method, the traditional framework selects the shortest path to the goal ignoring other agents. As argued before, this choice can lead to deadlocks despite the availability of the reactive collision avoidance method.

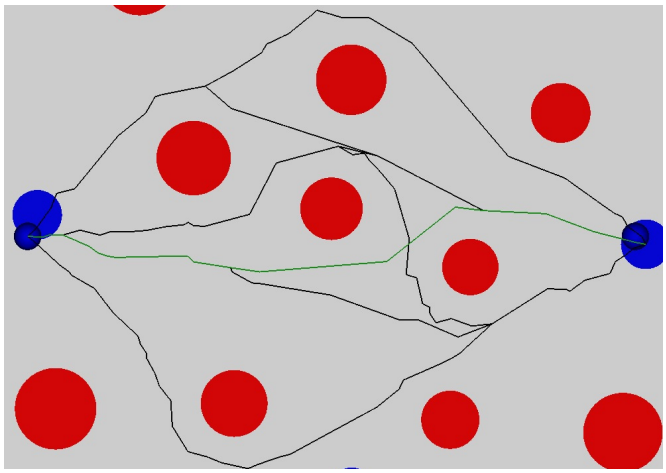


Figure 3: An example of a set of paths belonging in different homotopic classes for the right-side agent, bringing it to the left-side of the environment.

K-best Paths from Different Homotopic Classes

Rather than simply selecting the greedy path at each planning cycle, one alternative is for each agent to consider a diverse set of paths which bring the agent to its goal. This work accomplishes this by restricting the set of paths generated, such that they all must belong to different homotopic classes [Bhattacharya, Likhachev, and Kumar2011]. When considering 2D problems, trajectories are in different homotopy classes when the area between them contains an obstacle. A complete definition for homotopies can be found in the related literature [Hatcher2002].

By ignoring paths that loop around obstacles, the set of non-homotopic paths describes all of the shortest-length paths that bring the agent from its current position to the goal. These computations take place over an underlying roadmap, and use a set of search-based primitives. An example of the resulting set of computed paths in a simulated environment is shown in Figure 3.

The "k-best" strategy therefore corresponds to the following: at each replanning cycle, agents compute a set of k paths belonging to k different homotopies and select a single path from this set as the agent's action. Considering such a set of actions, instead of just the shortest path, provides the agent with more choices. It is then possible to select a path, not just greedily in terms of its length, but also in terms of its interactions with neighbors.

Minimizing Interaction Cost

The question now arises on how agents can differentiate among a set $\mathcal{S}$ of available paths from different homotopic classes in order to select motions that will allow the agents to make progress towards their goals. To describe the process employed by this work, consider the situation depicted in Figure 1, where agent A has actions a_1 to move through the lower corridor and a_2 to move through the upper corridor

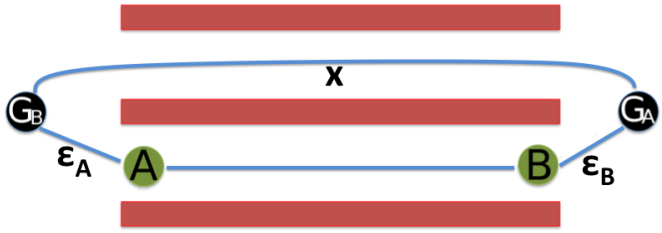


Figure 4: For the base case: for both of the robots the length to backtrack out of the current corridor and move to the other corridor is ϵ_A and ϵ_B for robots A and B respectively, while remaining in the current corridor reduces path length by ϵ_A and ϵ_B respectively.

towards its goal. These actions correspond to two solutions of the homotopic computation described in the previous section, regardless of the current configuration of the robot q_A. Similarly, agent B has choices b_1 and b_2.

Then the question is how costs $C(a_1)$, $C(a_2)$, $C(b_1)$, $C(b_2)$ can be computed appropriately, and in a decentralized manner, so that in any situation the two agents will decide to follow different corridors when they try to select the action with minimum cost. The most conflicted situation occurs when both robots are already following the path down the same corridor. Without loss of generality set both agents to be inside corridor 1, i.e. the lower corridor.

Assume that the goals for the agents are symmetrically placed at the end of each side of the corridor. Then the shortest path between the two goal points through the corridors is x, as illustrated in Figure 4. If the corridors are too narrow, then the paths will go through the current configurations of robots A and B. Assume that the distance between the goal G_B and q_A is ϵ_A and the distance between the goal G_A and q_B is ϵ_B along the path that goes through corridor 1. Then the lengths of the shortest paths for the robots to reach their goals via the corresponding homotopic paths can be computed as follows:

$$P_1^A = x - \epsilon_A, \quad P_2^A = x + \epsilon_A$$

$$P_1^B = x - \epsilon_B, \quad P_2^B = x + \epsilon_B$$

Where P_i^X corresponds to the length of the shortest path for robot X from its current configuration q_X to its goal G_X via corridor i.

The proposed approach also considers an interaction cost along each action for every agent. The interaction cost of an action is 0 if there is no other agent occupying the corresponding path given the latest observation. If there is an agent occupying the path, then the interaction cost is computed as follows:

$$I_i^A = 1 - \frac{\text{distance between A and B along } \pi_i}{\text{length of } \pi_i} \quad (1)$$

The reasoning behind this definition is that agents closer to the current position of an agent should incur a higher interaction cost. Then for the above scenario the interaction costs are:

$$I_1^A = \frac{\epsilon_B}{x - \epsilon_A}, \quad I_2^A = 0$$

$$I_1^B = \frac{\epsilon_A}{x - \epsilon_B}, \quad I_2^B = 0$$

Then the proposed cost function for actions is the following:

$$C_i^X = P_i^X(1 + 2 \cdot I_i^X) \quad (2)$$

which translates to the following cost in the above scenario:

$$C_1^A = x - \epsilon_A + 2 \cdot \epsilon_B, \quad C_2^A = x + \epsilon_A$$

$$C_1^B = x - \epsilon_B + 2 \cdot \epsilon_A, \quad C_2^B = x + \epsilon_B$$

Then, note that in order for A to select action 1 it has to be the case that:

$$C_1^A = x - \epsilon_A + 2 \cdot \epsilon_B < x + \epsilon_A = C_2^A \Rightarrow$$

$$A \text{ selects corridor 1 iff: } \epsilon_B < \epsilon_A \quad (3)$$

Similarly for robot B to select action 1 it has to be the case that:

$$C_1^B = x - \epsilon_B + 2 \cdot \epsilon_A < x + \epsilon_B = C_2^A \Rightarrow$$

$$B \text{ selects corridor 1 iff: } \epsilon_A < \epsilon_B \quad (4)$$

From Eqs. 3 and 4 it becomes apparent that the agents are not able to simultaneously pick the same corridor given the above definitions for the interaction cost and the overall cost functions. The agent who is farther away from its goal will have to pick the other homotopic class.

It is easy to check that if the agents had picked different corridors to enter then they would have stuck with their original choices as they would have incurred no interaction cost along the corridors that they would be moving. Similar reasoning can take place for the case that the environment has multiple corridors or the environment is the same and there are three agents A, B and C that are competing for the same corridor. Without loss of generality, if one assumes that in this case B is in the middle of the other two agents and C wants to move towards the same direction as B, there are two possible outcomes given the above definitions for the interaction cost and depending on the exact distances the three agents have traveled down the corridor:

- either A and C will be forced to change homotopic class and B continues down the same corridor,
- or A will continue moving along the same corridor and both B and C are forced to move to another corridor.

This means that C, which has the maximum number of conflicting agents along its path, will never decide to continue moving along the same corridor and the choice of A and B is forced to be different as in the case of two agents competing for the same corridor.

The entire above discussion was based on the assumption that the goal locations of the two agents were symmetrical relative to the corridors, i.e., the length of the path connecting the agents that goes through corridor 1 is the same as the length of the path through corridor 2. If the goals are not symmetrical, then instead of a common path length of x, the initial path costs P_i^X should include different lengths x_1 and x_2 for the connections of the goals via corridor 1 and 2 respectively. Then, the cost of actions should be defined in a general manner: $C_i^X = P_i^X(1 + \alpha \cdot I_i^X)$ for

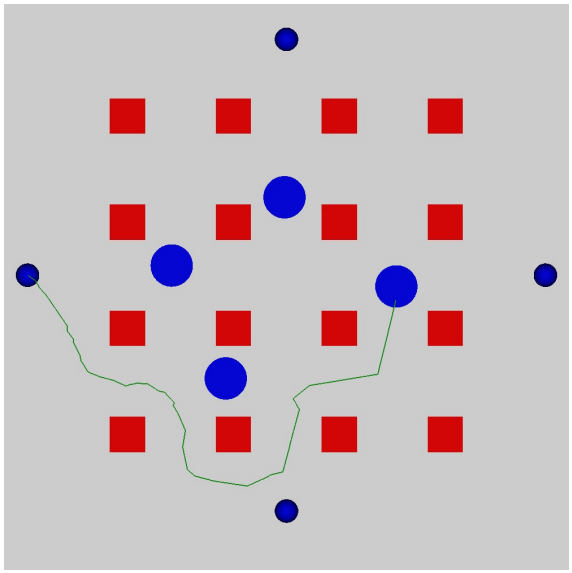


Figure 5: An example of a "minimum-conflict" path computed for the right-most agent for a goal on the left side of the image. In this situation the shortest path that does not conflict with any of the other agents is returned. In simulations with a larger number of agents, e.g., 32 in the above scene, it is typically the case that the "minimum-conflict" path still intersects with neighboring agents.

a constant α, which will depend on the relative difference $\Delta x = |x_1 - x_2|$. This is information, however, that is not available to the robots, since it requires knowledge of the goals for the other agents.

In practice, using the value $\alpha = 2$ as was defined originally in this section, results in good performance in the classification of different homotopic paths in terms of their interaction cost. So, in the context of the replanning framework in order to replace the greedy choice, a "k-best" choice is the following:

- Use a homotopy computation algorithm to extract the k-shortest paths that belong to k different homotopic classes.
- For each one of these paths, compute their costs according to Eq. 2, where the interaction cost is computed according to Eq. 1.
- Return the action with minimum cost.

The action with minimum cost both minimizes distance from the goal as well as interaction with other agents. The above "k-best" strategy is superior to the "greedy" strategy of always selecting the shortest path, since it allows multiple alternative choices to the robot and considers interactions with neighbors given the reasoning that was presented here for the basic "corridor" challenge under the assumption that the goals of the two agents are symmetric.

Minimum Conflict (MC) Path

The "k-best" strategy does not result in deadlocks in simulations as long as the number of simple homotopic classes does not significantly exceed k, where simple homotopic classes correspond to those that do not include loops. When this property is true, then the consideration of interaction costs with other agents, as described in the previous section, results in the selection of homotopy classes which allow the team to make progress overall. Even in relatively simplistic scenes, however, the number of homotopic classes required to satisfy this condition can quickly become large. This introduces a computational challenge, since the $k+1$ homotopy class corresponds to a longer path, which translates to a longer search time on the underlying roadmap. To keep the proposed method effective, however, it is important to keep a small planning cycle and perform each path computation as fast as possible.

In order to address this issue, the value of k is kept relatively small, and to accommodate the potential lack of a desirable path, the current work proposes that the "minimum-conflict path" should always be included as an available action to the agents. To compute such a path, each agent $\mathcal{A}_i$ considers the current set of configurations for the agents it can observe: $\{q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_m\}$. For each one of those configurations q_j, agent $\mathcal{A}_i$ marks edges in the roadmap that intersect q_j. Edges that are marked then have their weights inflated by a very large amount, effectively removing it from consideration during the heuristic search to find the shortest path on the roadmap from q_i to q_i^G. This means that the heuristic search process will first return the shortest path that does not collide with any agents. If no such path exists, then one which collides only with one agent will be returned and so on.

The inclusion of such paths in the set of available strategies results in methodologies that always solve challenges where the "greedy" or the "k-best" method failed. Interestingly, a strategy which only considers the "minimum conflict" action, constructed at each replanning cycle using the process described above, is also able to always solve all the challenges considered.

Even so, the resulting paths may not be as desirable when all the agents follow their minimum conflict action. As shown in Figure 5, this action may be significantly longer than the shortest path to the goal. Thus, it is interesting to consider the combination of the "k-best" strategy with the "minimum conflict" one. In this case, the process works as follows:

- Use a homotopy computation algorithm to extract the k-shortest paths that belong to k different homotopic classes.
- Compute the "minimum-conflict" action.
- For each one of the above paths, compute their costs according to Eq. 2, where the interaction cost is computed according to Eq. 1.
- Return the action with minimum cost.

This "deterministic" approach for combining the agent's greedy choices, i.e., k-shortest paths, and the safe choice, i.e., minimum conflict, takes again advantage of the process described in the previous section for evaluating a weighted cost of path length and interaction cost. It allows the agent to sometimes make the greedy choice and select one of the

shortest paths, even if they conflict with other agents, as long as these paths are significantly shorter than the minimum conflict path and do not overlap with other agents in a short time period.

A Probabilistic Selection Strategy

To allow some adaptability to varying conditions, this work considers an online learning method to compute a non-deterministic policy for selecting the appropriate strategy out of the following: (a) the "minimum conflict", i.e., the one that returns a path with the minimum number of collisions with other agents given their current configuration and among these paths, the one with the smallest length, and (b) a greedy strategy, where we have considered two versions:

- Always return the shortest path ignoring other agents and
- Return the action selected by the "k-best" strategy.

The idea is that the probabilistic selection strategy will learn during the execution of a path whether it is better to play the "minimum conflict" strategy or the greedy alternative, given the cost that it experiences for the outcomes of these strategies over time.

The learning algorithm used is the Polynomial Weights method, which is following the principle of regret minimization [Littlestone and Warmuth1994, Nisan et al.2007]. It begins by assigning uniform weights on the two strategies: $w^{min\_conflict} = w^{greedy} = 1$. Then, when the agent must choose an action, one of the strategies is chosen at random proportionally to their weights, i.e.,

$$Pr(\text{"minimum-conflict"}) = \frac{w^{min\_conflict}}{w^{min\_conflict} + w^{greedy}}$$

During each planning cycle, the method updates these weights by calculating a loss value for each one of them: $l^{min\_conflict}, l^{greedy}$, in hindsight, i.e., assuming that all the other agents would have acted the same way, the method computes a value that corresponds to the regret of choosing that value. Given the other agents' motion, one of the two pure strategies would have performed better. This action causes low regret and its weight is not reduced, while the worse performing strategy incurs regret, and thus receives a lowered weight. The implementation of the Polynomial Weights algorithm in the context of this challenge computes loss as follows:

$$l_i = \frac{C_i - min_i(C_i)}{max_i(C_i) - min_i(C_i)}$$

Where again the term C_i corresponds to the weighted cost computed according to Eq. 2. The weights are then updated according to the following rule and the computed loss value:

$$w_i = w_i \cdot (1 - \eta \cdot l_i)$$

This means that the action with the highest weighted cost in hindsight gets its weight reduced by η, while the other action is not penalized. A value of $\eta = 0.2$ was used for the simulations presented here.

The Polynomial Weights method has several advantages. First, it does not require knowledge of the other agent's utilities and requires no information to be passed from the other agents. Furthermore, as the weights are learned, the expected utility is guaranteed to be within a bound of the best pure strategy [Littlestone and Warmuth1994, Nisan et al.2007]. Lastly, it allows a high degree of adaptability to changing conditions, as large regret costs will be quickly accumulated for choosing a sub-optimal strategy.

Simulations

Each of the strategies presented in the previous sections,Greedy (`Greedy`), k-best (`KBest`), minimum-conflict (`Min Conf`), deterministic (`Determ`) (i.e., combination of "minimum-conflict" with "k-best"), Polynomial-Weights Greedy (`PWGreedy`), and Polynomial-Weights Best (`PWBest`), was evaluated experimentally in simulation.

The experiments were run in a computing cluster, where each agent was allowed access to a single computing core on an Intel Xeon E5-4650 2.70GHz, and given 1 GB of memory. This was done to simulate the fact that each agent represented a separate robot, so there was no competition between agents for computing resources. Each experiment had a homogeneous setup of agents, i.e. every robot ran the same strategy, within a variety of scenes, such as a grid and a random obstacle environment as shown in Fig. 6.

The following metrics were used to evaluate each strategy's performance:

- average completion time in seconds for all agents,
- average length of the solution path for all agents

Evaluating the average experimental solution time provides a good measure of the performance of the method, as it directly indicates how much progress agents are making towards their goals. The purpose of examining the average path length is to have some measurement of how much "effort" an agent must spend to achieve its desired solution time.

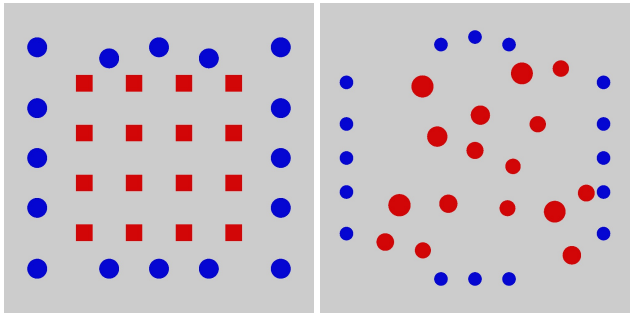


Figure 6: The environments used to evaluate the proposed methods. The blue disks are the agent's initial positions, when they are not randomized.

Corridor Experiments: Evaluating Validity

The experiments begin with a simple corridor setup, with only two agents attempting to reach opposite sides of the corridor. The purpose of such a simple setup is to find whether the proposed methods, including the `Greedy` approach, are able to solve simple congestion problems. The

results are averaged over 5 runs, with the average path lengths and solution times shown in Figure 7.

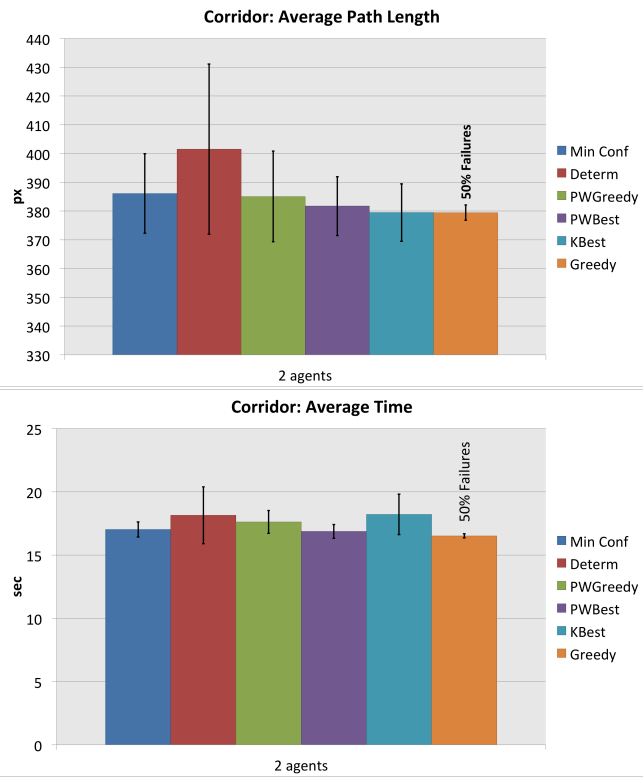


Figure 7: Results for testing the validity of the approaches in the corridor environment as shown in Figure 6.

Although `Greedy` always had the lowest averages, it failed to solve even a simple deconfliction problem such as this 50% of the time. This is again due to the fact that no other paths are considered by the agent. All of the other strategies were able to solve the corridor problem without a single failure.

Evaluating Performance

The performance of the four methods is evaluated using two environments, the grid environment and a forest-like environment as shown in Fig. 6. Since the `Greedy` strategy failed to consistently solve the corridor problem, it is omitted from the rest of the experiments. An important observation of the `KBest` strategy is for small values of k, and for large numbers of homotopy classes, it is possible for the strategy to become deadlocked/livelocked. Such was the case in the grid and forest environments, so accordingly the `KBest` strategy is no longer considered in further experiments.

Agents are given a pseudo-random start location, and a fixed goal location, with the intention of having agents swap locations with one another, which promotes conflicts and congestion in the environment. The results are averaged over 5 runs and presented in Fig. 8 (for the grid environment) and Fig. 9 (forest).

The deterministic approach, `Determ`, which considers

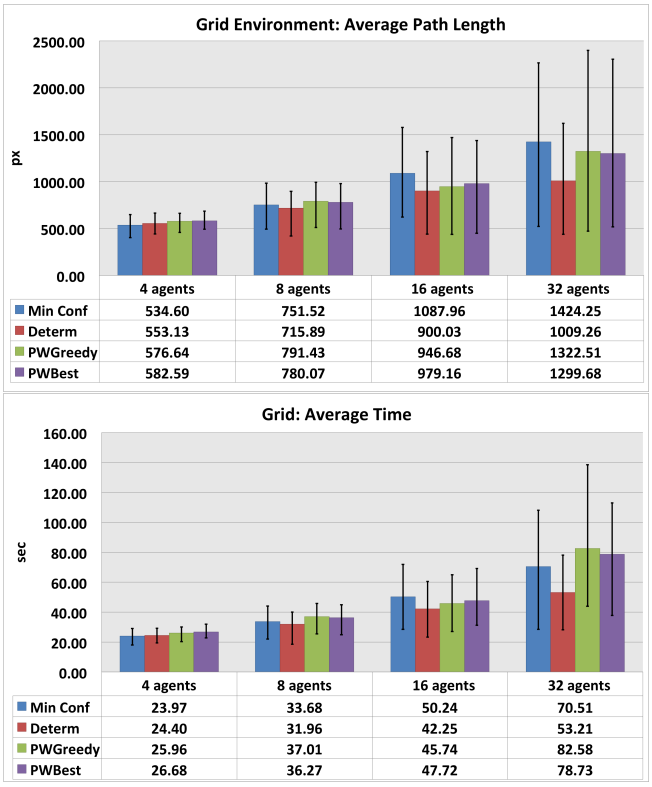


Figure 8: Results for randomly selected starting positions in the grid environment. Data is shown for increasing number of agents in the same environment, where error bars signify the variance over all simulations.

both the "minimum-conflict" and the "k-best" strategies, always selects the action that minimizes the proposed interaction cost. In the Grid environment, `Determ` outperformed the other approaches. In the random obstacle forest scene, however, the methods seem to be competitive. The explanation for this is that the random placement of obstacles, combined with a larger workspace, does not cause a constrained enough environment, hence there are not frequent conflicts between agents. This allows agents to consider a larger set of possible actions that are conflict-free, so each of the strategies presented can provide an equivalent solution quality.

Grid Experiments: Evaluating Scalability

In these experiments the start location of the agents were set to be symmetrical, so as to promote conflicts and congestion quickly. The results are averaged over 15 different runs and are shown in Figure 10. The purpose of this set of experiments was to evaluate the scalability of the adaptive-strategies, `PWGreedy` and `PWBest`, as both of these approaches utilize the other deconfliction methods, and are consequently the most computationally complex.

The results show that for increasingly larger number of agents, the average solution time and the average path lengths for the methods scales sublinearly.

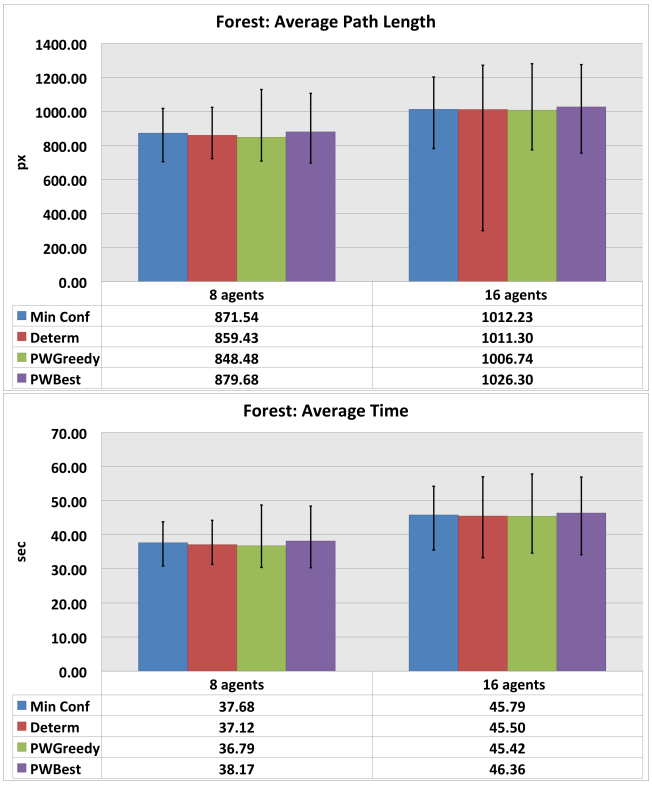


Figure 9: Results for randomly selected starting positions in the random obstacle, forest environment.

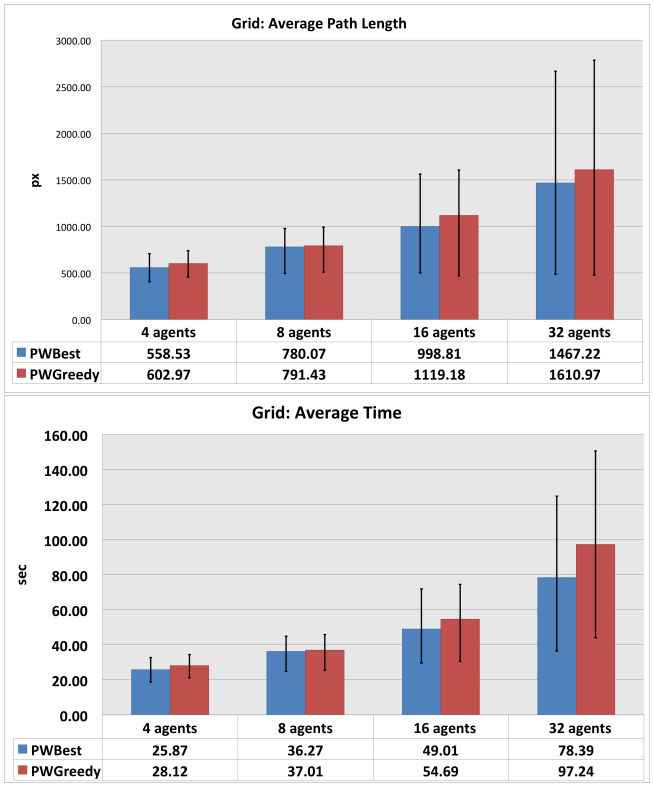


Figure 10: The average path length and average time to finish for simulations using the polynomial weight with greedy (`PWGreedy`) and with k-best selection (`PWBest`).

Heterogeneous Setups

A set of experiments was conducted among heterogeneous agents in the grid environment, where 7 agents were assigned the "minimum-conflict" strategy and 1 agent was assigned the PWBest strategy. The idea here was to examine the probabilistic learning algorithm, PWBest, and see if it was able to adapt its weights according to the strategies the other agents were playing. Interestingly, over a course of 5 separate runs, PWBest selected the "k-best" strategy 65% of the time on average. Since the 'minimum-conflict" agents were actively attempting to avoid interaction with other agents, it makes sense that the PWBest agent is able to be more "greedy" in its selection of paths.

Carrying on with this line of thought, another set of experiments was run where 4 agents were given the pure "greedy" strategy, and the other 4 agents ran PWBest. In this case, the PWBest agents adapted and chose to select the "k-best" strategy only 41% of the time. Since the 4 purely greedy agents caused a deadlock in the center of the environment, the PWBest agents had to adapt and select the safer "minimum-conflict" strategy more often. Together these results seem to show promise for the adaptability of the learning strategy, as well as motivating its use over the "deterministic" strategy.

A video of the experiments can be found at:

`http://www.cs.rutgers.edu/~kb572/videos/icaps_PlanRobWorkshop_2014.mp4`

Discussion

The proposed framework brings together path planning primitives, such as search-based methods for computing paths in different homotopic classes [Bhattacharya, Likhachev, and Kumar2011] and sampling-based motion planners for computing roadmaps [Karaman and Frazzoli2011], reactive obstacle avoidance methods [van den Berg, Lin, and Manocha2008, Snape et al.2011] as well as game theoretic and learning tools [Nisan et al.2007] to provide an algorithmic framework capable of computing acceptable solutions to motion coordination challenges in a decentralized, communication-less way.

Some interesting directions for this work include: removing the "garage" assumption from the framework - this would require agents to continue reasoning about their observed states, potentially adapting a "passive" mode to more easily allow other agents through their goal positions; extensively evaluate the adaptive methods in a larger set of heterogeneous setups, as well as imposing a stricter sensing range on the agents; and analyzing the conditions under which the current framework is able to guarantee that the robots are free of deadlocks and livelocks, using tools that have been developed towards this direction [Knepper and Rus2013]. Currently, the work is in the process of being evaluated on real systems in a construction challenge, with the eventual hope of running experiments including humans.

References

Bekris, K. E.; Grady, D. K.; Moll, M.; and Kavraki, L. E. 2012. Safe Distributed Motion Coordination For Second-Order Systems With Different Planning Cycles. *International Journal of Robotics Research (IJRR)* 31(2).

Bennewitz, M.; Burgard, W.; Cielniak, G.; and Thrun, S. 2005. Learning Motion Patters of People for Compliant Robot Motion. *International Journal of Robotics Research (IJRR)* 24(1):31–48.

Bhattacharya, S.; Kumar, V.; and Likhachev, M. 2010. Search-based Path Planning with Homotopy Class Constraints. In *Third Annual Symposium on Combinatorial Search*.

Bhattacharya, S.; Likhachev, M.; and Kumar, V. 2011. Identification and Representation of Homotopy Classes of Trajectories for Search-based Path Planning in 3D. In *Proc. of Robotics: Science and Systems*.

Excelente-Toledo, C. B., and Jennings, N. R. 2004. The dynamic selection of coordination mechanisms. *Autonomous Agents and Multi-Agent Systems* 9(1-2):55–85.

Filiot, E.; Le Gall, T.; and Raskin, J.-F. 2010. Iterated regret minimization in game graphs. In *Mathematical Foundations of Computer Science 2010*. Springer. 342–354.

Fiorini, P., and Shiller, Z. 1998. Motion planning in dynamic environments using velocity obstacles. *Int. Journal of Robotics Research* 17(7).

Fraichard, T., and Delsart, V. 2009. Navigating Dynamic Environments with Trajectory Deformation. *Journal of Computing and Information Technology* 17(1).

Green, C., and Kelly, A. 2007. Toward Optimal Sampling In the Space of Paths. In *International Symposium on Robotics Research (ISRR)*.

Hatcher, A. 2002. *Algebraic Topology*. Cambridge University Press.

Hauser, K. 2011. Adaptive time stepping in real-time motion planning. In *Algorithmic Foundations of Robotics IX*. Springer. 139–155.

Hauser, K. 2013. Minimum Constraint Displacement Motion Planning. In *Proc. of Robotics: Science and Systems*.

Henry, P.; Vollmer, C.; Ferris, B.; and Fox, D. 2010. Learning to Navigate Through Crowded Environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Jaillet, L., and Siméon, T. 2006. Path Deformation Roadmaps. In *Workhop on the Algorithmic Foundations of Robotics (WAFR)*.

Kaminka, G. A.; Erusalimchik, D.; and Kraus, S. 2010. Adaptive multi-robot coordination: A game-theoretic perspective. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 328–334. IEEE.

Karaman, S., and Frazzoli, E. 2011. Sampling-based Algorithms for Optimal Motion Planning. In *IJRR*.

Karamouzas, I.; Geraerts, R.; and Overmars, M. 2009. Indicative Routes for Path Planning and Crowd Simulation. In *The 4^{th} Intern. Conference on the Foundations of Digital Games (FDG)*, number 113-120.

Knepper, R. A., and Mason, M. T. 2009. Path Diversity is Only Part of the Problem. In *Proc. of the IEEE Intern. Conf. on Robotics and Automation (ICRA)*.

Knepper, R. A., and Rus, D. 2012. Pedestrian-Inspired Sampling-based Multi-Robot Collision Avoidance. In *Proc. of the International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 94–100. Paris, France: IEEE.

Knepper, R. A., and Rus, D. 2013. On the Completeness of Ensembles of Motion Planners for Decentralized Planning. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Large, F.; Vasquez, D.; Fraichard, T.; and Laugier, C. 2004. Avoiding Cars and Pedestrians Using Velocity Obstacles and Motion Prediction. In *IEEE Intelligent Vehicles Symposium*.

Littlestone, N., and Warmuth, M. K. 1994. The Weighted Majority Algorithm. *Information and Computation* 108:212–261.

Nisan, N.; Roughgarden, T.; Tardos, E.; and Vazirani, V. V. 2007. *Algorithmic game theory*. Cambridge University Press.

Petti, S., and Fraichard, T. 2005. Partial Motion Planning Framework for Reactive Planning within Dynamic Environments. In *ICINCO*, 199–204.

Qutub, S.; Alami, R.; and Ingrand, F. 1997. How to Solve Deadlock Situations within the Plan-Merging paradigm for Multi-Robot Cooperation. In *Proc. of the Inter. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, 1610–1615.

Shi, D.; Collins, E. G.; Donate, A.; Liu, X.; Goldiez, B.; and Dunlap, D. 2008. Human-aware Robot Motion Planning with Velocity Constraints. In *IEEE International Symposium on Collaborative Technologies and Systems*, 490–497.

Sisbot, E. A.; Marin-Urias, L. F.; Alami, R.; and Siméon, T. 2007. A Human-aware Mobile Robot Motion Planner. *IEEE Transactions on Robotics* 23(5):874–883.

Snape, J.; van Den Berg, J.; Guy, S.; and Manocha, D. 2011. The Hybrid Reciprocal Velocity Obstacle. *IEEE Transactions on Robotics* 27(4):696–706.

Stone, P.; Kaminka, G. A.; and Rosenschein, J. S. 2010. Leading a best-response teammate in an ad hoc team. In *Agent-Mediated Electronic Commerce. Designing Trading Strategies and Mechanisms for Electronic Markets*. Springer. 132–146.

Thompson, S.; Horiuchi, T.; and Kagami, S. 2009. A Probabilistic model of Human Motion and Navigation Intent for Mobile Robot Path Planning. In *Proc. of the 4^{th} International Conference on Autonomous Robots and Agents*.

Trautman, P., and Krause, A. 2010. Unfreezing the Robot: Navigation in Dense, Interacting Crowds. In *Proc. of the IEEE Intern. Conf. on Intelligent Robots and Systems (IROS)*.

van Den Berg, J.; Patil, S.; Sewall, J.; Manocha, D.; and Lin, M. 2008. Interactive Navigation of Individual Agents in Crowded Environments. In *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*.

van den Berg, J.; Snape, J.; Guy, S.; and Manocha, D. 2011. Reciprocal Collision Avoidance with Acceleration-Velocity Obstacles. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*.

van den Berg, J.; Lin, M.; and Manocha, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Wunder, M.; Kaisers, M.; Yaros, J. R.; and Littman, M. 2011. Using iterated reasoning to predict opponent strategies. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 593–600. International Foundation for Autonomous Agents and Multiagent Systems.

Ziebart, B. D.; Ratliff, N.; Gallagher, G.; Mertz, C.; Peterson, K.; Bagnell, J. A.; Hebert, M.; Dey, A.; and Srinivasa, S. 2009. Planning-based Prediction for Pedestrians. In *Proc. of the IEEE Intern. Conf. on Intelligent Robots and Systems (IROS)*.

Efficient and Smooth RRT Motion Planning Using a Novel Extend Function for Wheeled Mobile Robots

Luigi Palmieri and **Kai O. Arras**
Social Robotics Laboratory
Dept. of Computer Science
University of Freiburg
Germany

Abstract

In this paper we introduce a novel RRT extend function for wheeled mobile robots. The approach computes closed-loop forward simulations based on the kinematic model of the robot and enables the planner to efficiently generate smooth and feasible paths that connect any pairs of states. We extend the control law of an existing discontinuous state feedback controller to make it usable as an RRT extend function and prove that all relevant stability properties are retained. We study the properties of the new approach as extender for RRT and RRT* and compare it systematically to a spline-based approach and a large and small set of motion primitives. The results show that our approach generally produces smoother paths to the goal in less time with smaller trees. For RRT*, the approach produces also the shortest paths and achieves the lowest cost solutions when given more planning time.

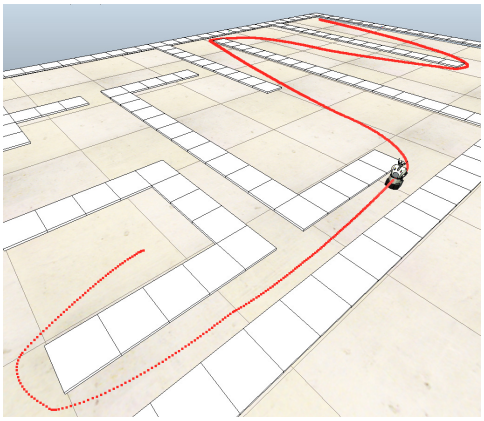


Figure 1: A smooth motion planning solution computed by our new extender.

1 Introduction

Planning with rapidly-exploring random trees (RRT) (LaValle and Kuffner 1999) has become a popular approach to robot motion planning. RRT planners are single-query sampling-based planners that grow a tree of configurations to eventually cover the entire state space. A probabilistically optimal RRT variant named RRT* has been introduced by Karaman and Frazzoli (Karaman and Frazzoli 2010). RRT* trees grow based on the notion of a cost: under the assumptions given in (Karaman and Frazzoli 2011) the solution converges to the optimum as the number of samples approaches infinity.

For robots with kinematic or kinodynamic constraints, the *extend function*, the function that grows the tree by finding collision-free trajectories to new sampled configurations, becomes a key component. Its task is to connect any pair of states under differential constraints which represents by itself a local planning problem also referred to as the *two-point boundary value problem*.

Here, we consider the motion planning problem for non-holonomic wheeled robots in 2D with the goal of particularly smooth and natural real-time motion generation for robots in human environments. To this end, we propose a new extend function for RRT and RRT* which shall enable the planner to efficiently generate smooth paths. Previously used extend functions include motion primitives (LaValle and Kuffner 1999), (Frazzoli, Dahleh, and Feron 2005), (Kalisiak and van de Panne 2006), (Kalisiak and van de Panne 2007), optimal controllers (Perez et al. 2012) (Webb and van den Berg 2013), shooting methods (Hwan Jeon, Karaman, and Frazzoli 2011), splines (Yang et al. 2014), and closed-loop controllers (Kuwata et al. 2009).

Motion primitives have originally been proposed for RRT-based planning under differential constraints. LaValle and Kuffner (LaValle and Kuffner 1999) implement the extend function as a forward simulation of a set of predefined discretized controls, so called motion primitives. The approach satisfies the constraints, is efficient to compute and easy to implement: the tree is extended with the primitive that is found to come closest to the new sampled configuration $\mathbf{x}_{new}$. However, the method has several shortcomings: it does not fully solve the two-point boundary value problem as the orientation of $\mathbf{x}_{new}$ is ignored, the extension of the tree even by the closest motion primitive may still be far-off from

$\mathbf{x}_{new}$, and the concatenation of primitives may lead to sequences of discontinuous inputs and non-smooth trajectories. The last point was addressed by Frazzoli *et al.* (Frazzoli, Dahleh, and Feron 2005) who propose a finite-state machine called a Maneuver Automaton to allow correct (and thereby smooth) concatenation of motion primitives to complex motion trajectories. However, its use in RRT-based planning has not been studied.

Recently, Perez *et al.* (Perez et al. 2012) use an optimal infinite-horizon LQR controller to connect pairs of states. The method linearizes the domain dynamics locally, which is interesting from an efficiency point of view, but will in general not reach the target state exactly. Webb and van den Berg (Webb and van den Berg 2013) use a finite-horizon optimal controller as local planner. They can optimize a certain class of cost functions to trade off between time and control effort. Goretkin *et al.* (Goretkin et al. 2013) use a finite-horizon LQR controller extended to affine systems. They can generically extend the algorithm to nonlinear systems by linearizing the dynamics at vertices in the tree: the obtained approximations are in general affine. Although optimal control techniques may produce high-quality solutions to the two-point boundary value problem, they typically suffer from high computational costs and numerical issues that can make them unsuitable for motion planning in real-time.

Hwan Jeon *et al.* (Hwan Jeon, Karaman, and Frazzoli 2011) use the shooting method to numerically solve the two-point boundary value problem to obtain an extend function for RRT*. The method allows for time-optimal maneuvers of a high-speed off-road vehicle. As with optimal control techniques, shooting methods may have issues with numerical stability and computational costs for our application.

In a recent work, Yang *et al.* (Yang et al. 2014) use splines as RRT extend function. The authors take cubic Bézier splines that guarantee curvature continuity of paths and are able to satisfy upper-bounded curvature constraints. With our goal of smooth and natural motion generation, we consider splines to be a potentially interesting approach and include a spline-based extend function into our experimental comparison. However, instead of cubic Bézier splines which are limited to curves with continuous curvature, we will use η^3 splines, introduced by Piazzi *et al.* (Piazzi, Bianco, and Romano 2007) that produce curves with a continuous derivative of the curvature, therefore generating even smoother paths than cubic Bézier splines.

Kuwata *et al.* (Kuwata et al. 2009) introduce closed-loop RRT (CL-RRT), a modified RRT for real-time local lane following with a car using an extend function based on a closed-loop model. Given a sampled control input, the method runs a forward simulation using the vehicle and controller models to predict and then evaluate extend trajectories.

The contribution of our work is as follows:

- We propose an extender based on closed-loop predictions for a non-holonomic wheeled mobile robot. It efficiently solves the two-point boundary value problem by exponentially converging to the goal state from any start state. We extend the control law of the original approach by Astolfi (Astolfi 1999) with a term that leads to quasi-constant path velocities along local path concatenations – a key ability for RRT extend functions. We also prove the relevant stability properties under our modification.
- We systematically compare our approach to two alternative extenders, namely motion primitives (two sets of different size) and splines. The experiments demonstrate that our approach outperforms both methods in many relevant metrics: smoother paths and shorter planning time (with RRT), shorter paths (with RRT*), and significantly smaller trees (both). We also find that our method can benefit most from the incremental path improvement ability of RRT* resulting in the lowest cost solutions when given more planning time.
- To the best of our knowledge, this paper presents the first systematic study of the impact of different extend functions on RRT and RRT* performance and path quality. Its necessity is corroborated by the significant variations of key metrics only caused by the use of different extend functions.

The paper is structured as follows: the next Section reviews the RRT algorithm and typical extend functions. In Section 3 we describe our approach which is then experimentally evaluated in Section 4. We discuss the results in Section 5, and Section 6 concludes the paper.

2 Rapidly Exploring Random Trees

We briefly review the RRT algorithm for planning under differential constraints. Let $\mathcal{X} \subset \mathbb{R}^d$ be the configuration space and $\mathcal{U} \subset \mathbb{R}^m$ the control space. A nonholonomic wheeled mobile robot can be described by a differential equation as

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \qquad \mathbf{x}(0) = \mathbf{x}_0 \tag{1}$$

where $\mathbf{x}(t) \in \mathcal{X}$, $\mathbf{u}(t) \in \mathcal{U}$, for all t, $\mathbf{x}_0 \in \mathcal{X}$ and f is a function describing the kinematics of the system. The RRT algorithm solves a feasible kinematic motion planning problem p: given an obstacle space $\mathcal{X}_{obs} \subset \mathcal{X}$, a free space $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$, an initial state $\mathbf{x}_{init} \in \mathcal{X}_{free}$ and a goal region $\mathcal{X}_{goal} \subset \mathcal{X}_{free}$, find a control $\mathbf{u}(t) \in \mathcal{U}$ with domain $[0, T]$, $T > 0$, such that the unique trajectory $\mathbf{x}(t)$ satisfies equation (1), is in the free space $\mathcal{X}_{free} \subseteq \mathcal{X}$ and goes from $\mathbf{x}_{init}$ to a goal $\mathbf{x}_{goal} \in \mathcal{X}_{goal}$. The RRT procedure is outlined in Algorithm 1.

2.1 Extend Function

The purpose of the extend function is to connect new states to the tree: it grows a branch from $\mathbf{x}_{near}$ toward $\mathbf{x}_{rand}$. The terminal state of the new branch, $\mathbf{x}_{new}$, may differ (largely) from $\mathbf{x}_{rand}$ depending on the extend function used. $\mathbf{x}_{new}$ is then added to the tree τ together

Algorithm 1 Rapidly-exploring Random Tree

function RRT($\mathbf{x}_{init}$, $\mathbf{x}_{goal}$)
$\tau.add\_vertex(\mathbf{x}_{init})$
while $k \leq K$ **do**
  $\mathbf{x}_{rand} \Leftarrow random\_state(\mathcal{X})$
  $\mathbf{x}_{near} \Leftarrow nearest\_neighbor(\tau, \mathbf{x}_{rand})$
  $\mathbf{x}_{new}, \mathbf{u}_{best} \Leftarrow extend(\mathbf{x}_{near}, \mathbf{x}_{rand})$
  $\tau.add\_vertex(\mathbf{x}_{new})$
  $\tau.add\_edge(\mathbf{x}_{near}, \mathbf{x}_{new}, \mathbf{u}_{best})$
  if $\mathbf{x}_{new} \in \mathcal{X}_{goal}$ **then**
    return $extract\_traj(\mathbf{x}_{new})$
  end if
end while
return *failure*

with the intermediate points of the new local path and the selected $\mathbf{u}$. The expansion fails if a collision along the path occurs.

We briefly review motion primitives, the originally proposed extenders as part of RRT. The approach is based on the forward propagation of a control input into a system simulator. Given an initial state $\mathbf{x}_0$, an integration time Δt, an integration time step t_s and a input $\mathbf{u}_i$ from a discrete set of controls $\mathcal{U} = \{\mathbf{u}_1, .., \mathbf{u}_m\}$, a trajectory $\mathbf{x}_i(t)$ is generated by numerically integrating Eq. (1)

$$\mathbf{x}_i(t) = \int_0^{\Delta t} f(\mathbf{x}_i(t), \mathbf{u}_i(t))\ dt + \mathbf{x}_0, \quad i = 1, ..., m. \quad (2)$$

All the controls in $\mathcal{U}$ are checked, and the one that brings the expansion closest to $\mathbf{x}_{rand}$ (according to a distance metric) is stored together with the associated local trajectory that will be added to the tree τ. To minimize the time needed to extend the tree, the motion primitives can be precomputed off-line.

An alternative approach to extending the tree is to employ a full-fetched local planner that generates trajectories $\mathbf{x}(t) \in \mathcal{X}_{free}$ and the corresponding continuous controls $\mathbf{u}(t)$.

2.2 RRT* Extend Function

The extension procedure in RRT* is more complex. It is based on the concept of *near neighbors*, the neighbors within a specified radius of a node. The first step of the extension is to connect a newly added vertex to its neighbor vertex with minimal cost. The next step is to rewire the tree: if the path from the newly created vertex to a near neighbor node has a lower cost than the near neighbor, then the parent of the near vertex is changed to the new vertex. Each time the algorithm attempts to connect two vertices a *steer function* is called for which RRT extend functions can be used.

3 The Approach: POSQ

The proposed extend function computes closed-loop forward simulations based on the kinematic model of a non-holonomic wheeled mobile robot. It generates the trajectory $\mathbf{x}(t)$ and controls $\mathbf{u}(t)$, $t \in [0, T]$, $T > 0$, that connect any given pair of poses. Thus, it solves the two-point boundary value problem for such kinematic systems, see Fig. 2. The tree is grown in the configuration space $\mathbb{R}^2 \times \mathbb{S}^1$ where each configuration $\mathbf{x}$ consists of the Cartesian position of the wheeled mobile robot and its orientation, i.e. (x, y, θ).

The approach, originally proposed by Astolfi (Astolfi 1999), solves the problem of exponential stabilization of the kinematic and dynamic model of the wheeled mobile robot. It is not an optimal controller but has provable local and global stability, a light-weight implementation, and generates smooth trajectories. We believe that for extend functions, optimality is less relevant than efficiency, smoothness and the ability to fully solve the two-point boundary value problem. This is particularly true for RRT* for non-holonomic dynamical systems (Karaman and Frazzoli 2013), where the steering function must fulfil the topological property (Laumond, Sekhavat, and Lamiraux 1998).

We briefly summarize the original approach (Astolfi 1999) and describe our extension in the next subsection. Let ρ be the Euclidean distance between the initial pose and the goal pose ($\mathbf{x}_{near}$ and $\mathbf{x}_{rand}$ in an RRT notation), ϕ the angle between the x-axis of the robot reference frame $\{X_R\}$ and the x-axis of the goal pose frame $\{X_G\}$, α the angle between the y-axis of the robot reference frame and the vector Z connecting the robot with the goal position, v the translational and ω the angular robot velocity (Fig. 3). Then, the method makes a Cartesian-to-polar coordinates transform to describe the kinematics using the open loop model

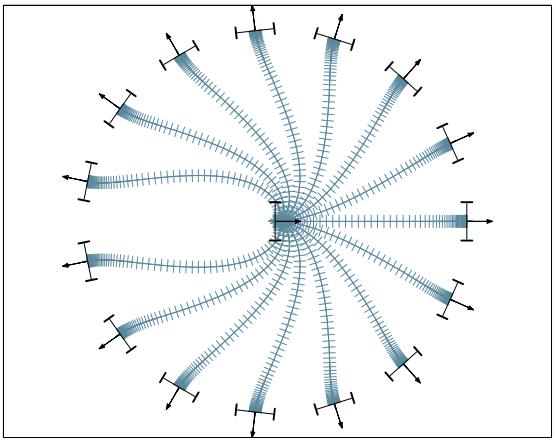


Figure 2: Trajectories of the controller when steering the robot from the center to the poses on the circle

$$\begin{aligned} \dot{\rho} &= -\cos\alpha\ v, \\ \dot{\alpha} &= \frac{\sin\alpha}{\rho} v - \omega, \\ \dot{\phi} &= -\omega, \end{aligned} \quad (3)$$

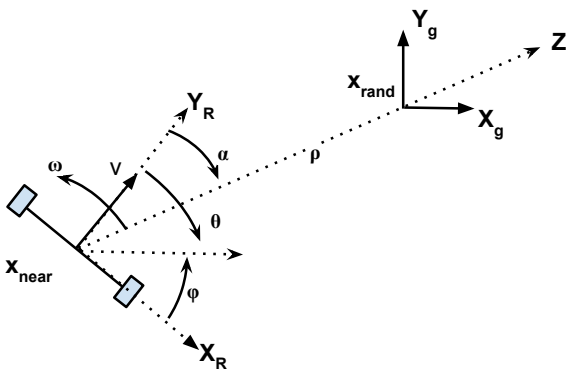


Figure 3: Notation and robot to goal relations

and the feedback law

$$\begin{aligned} v &= K_\rho \rho, \\ \omega &= K_\alpha \alpha + K_\phi \phi. \end{aligned} \tag{4}$$

As shown in (Astolfi 1999), this feedback law guarantees smooth trajectories without cusps.

3.1 Our control law

The original approach, however, generates trajectories of decaying forward velocity bringing the robot to a stop at each goal. The concatenation of such local paths would result in final paths of unnatural and slow movements.

Thus, we modify the feedback law so as to have quasi constant forward velocity at a desired maximum value across multiple expansions. We will prove that this modification retains local stability and that the robot's heading converges asymptotically to the desired equilibrium point.

Considering the open loop model in Eq. (3) obtained by the polar coordinate transform, we define the nonlinear feedback law

$$\begin{aligned} v &= K_\rho \tanh(K_v \rho), \\ \omega &= K_\alpha \alpha + K_\phi \phi \,. \end{aligned} \tag{5}$$

Substituting the control law (5) into the open loop model we obtain the following closed loop model

$$\begin{aligned} \dot{\rho} &= -K_\rho \cos\alpha \tanh(K_v \rho), \\ \dot{\alpha} &= K_\rho \frac{\sin\alpha}{\rho} \tanh(K_v \rho) - K_\alpha \alpha - K_\phi \phi, \\ \dot{\phi} &= -K_\alpha \alpha - K_\phi \phi \,. \end{aligned} \tag{6}$$

We now describe the conditions for which local stability holds and prove heading convergence.

Local Stability We can locally approximate the closed loop model (6) by

$$\begin{aligned} \dot{\rho} &= -K_\rho K_v \rho \\ \dot{\alpha} &= -(K_\alpha - K_\rho K_v)\alpha - K_\phi \phi \\ \dot{\phi} &= -K_\alpha \alpha - K_\phi \phi \end{aligned}$$

which is locally exponentially stable if and only if the eigenvalues of the matrix describing the linear approximation of the model have all negative real parts. For that, we need to have

$$\begin{aligned} K_v &> 0, \\ K_\rho &> 0, \\ K_\phi &< 0, \\ K_\alpha + K_\phi - K_\rho K_v &> 0 \,. \end{aligned} \tag{7}$$

Considering the closed loop model (6), assume $\alpha(0) \in]-\frac{\pi}{2}, \frac{\pi}{2}]$, and $\phi(t) \in\ ]n\pi, n\pi]$ for all t. Then, if

$$K_\alpha + 2nK_\phi - \frac{2}{\pi} K_\rho K_v > 0 \tag{8}$$

holds one has $\alpha(t) \in]-\frac{\pi}{2}, \frac{\pi}{2}]$ for all $t > 0$ which means that the robot trajectory will always stay in this region: we have defined a *trapping region*. Thus, together with the condition $K_\rho, K_v > 0$, the robot will move monotonically towards the origin.

Heading Convergence We want the robot to move towards to goal $\mathbf{x}_{rand}$ but notice in (6) that the goal position (the origin) can not be reached because ρ is a singularity point. Thus, we define an arbitrarily small number to which ρ converges, $\rho \to \epsilon$, with $\epsilon > 0,\ \rho > \epsilon$. Let us focus on the following reduced subsystem which describes how the orientation evolves

$$\begin{aligned} \dot{\alpha} &= -K_\alpha \alpha - K_\phi \phi + K_\rho \sin\alpha \frac{\tanh(K_v \rho)}{\rho}, \\ \dot{\phi} &= K_\alpha \alpha - K_\phi \phi \,. \end{aligned}$$

Given that $\dot{\rho}$ is strictly negative, we want to find the conditions for which the above vector field has a unique equilibrium point ($\alpha = 0,\ \phi = 0$) to which all trajectories converge asymptotically for all $\rho > \epsilon$. This is equivalent to minimizing the orientation error as well as stopping the robot at $\mathbf{x}_{new}$, ϵ meters away from $\mathbf{x}_{rand}$.

If we consider the candidate Lyapunov function

$$V(\alpha, \phi) = (-K_\alpha \alpha + K_\phi \phi)^2 + 2\ K_\phi K_\rho\ (\cos\alpha - 1) \tanh(K_v \rho), \tag{9}$$

we can show that

$$\dot{V}(\alpha, \phi) = 2\ K_\phi K_\rho \alpha\ \sin\alpha\ \tanh(K_v \rho) \left[K_\phi + K_\alpha - K_\rho \frac{\sin\alpha}{\alpha}\ \tanh(K_v \rho) \right]. \tag{10}$$

Given that conditions (7) and (8) hold and considering $n = 2$, V is positive and $\dot{V}$ is non-positive in all $S_2 = \left\{ (\alpha, \phi) \in R^2 \mid \alpha \in]-\frac{\pi}{2}, \frac{\pi}{2}],\ \phi \in (-2\pi, 2\pi] \right\}$.

It exists a positively invariant set

$$M = \left\{ (\alpha, \phi) \in S_2 |\ \ V \leq \frac{9}{4} k_\phi^2 \pi^2 - 2K_\phi K_\rho \right\}.$$

which is contained in S_2, and it contains $S_1 = \left\{ (\alpha, \phi) \in R^2 \mid \alpha \in]-\frac{\pi}{2}, \frac{\pi}{2}],\ \phi \in (-\pi, \pi] \right\}$. M contains only the equilibrium point ($\alpha = 0,\ \phi = 0$). Thus, all

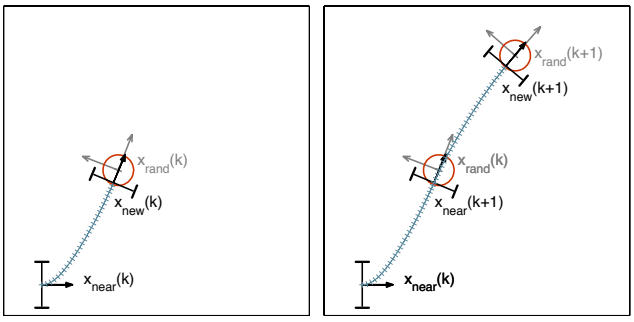


Figure 4: A simple two-step expansion example. With k being the time index of successive extensions, the proposed controller extends the tree from $\mathbf{x}_{near}(k)$ to $\mathbf{x}_{rand}(k)$ until the local trajectory enters the disk of radius γ at $\mathbf{x}_{new}(k)$. The procedure is repeated for $k+1$.

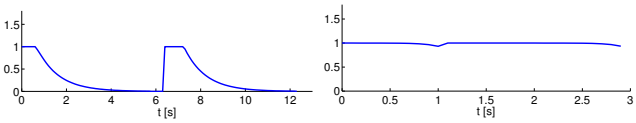


Figure 5: Translational robot velocity in [m/s] with the original control law from (Astolfi 1999) (left) and our control law (right) across the concatenation of two extensions. For the same two goal poses the new law allows for much faster movements.

trajectories starting in S_1 and contained in S_2 converge asymptotically to the origin according to the Poincare-Bendixson Theorem.

Notice that we are not solving a stabilization problem like in the original approach (Astolfi 1999). The new control law allows us, during expansion of the tree from $\mathbf{x}_{near}$ towards $\mathbf{x}_{rand}$, to minimize the error in orientation and stop when the local trajectory is close enough to the goal $\mathbf{x}_{rand}$. A $\gamma > 0$ threshold can be defined as minimum Euclidean distance that stops the expansion towards $\mathbf{x}_{rand}$. It is guaranteed that the terminal state $\mathbf{x}_{new}$ is not further away from $\mathbf{x}_{rand}$ than γ, which thus becomes a tunable error bound.The threshold γ can be seen as the radius of a circle centered at $\mathbf{x}_{rand}$, see Fig. 4. In practice, γ is chosen to be a few centimetres.

The new control law does not remove the velocity decay toward the goal but makes it significantly sharper. So sharp, that even small values for γ cause the decay to disappear and allow for quasi-constant forward velocities along the previously explained extension procedure. See Fig. 5 for a comparison. The method is named *POSQ* as it acts like a pose controller.

4 Experiments

In the experiments, we evaluate the new extend function and compare it to two alternative methods, namely motion primitives (two sets of different size) and splines. We quantify their impact on planning performance in terms of time, tree size and path quality in three different simulated environments. We use both RRT and RRT* as planning algorithms.

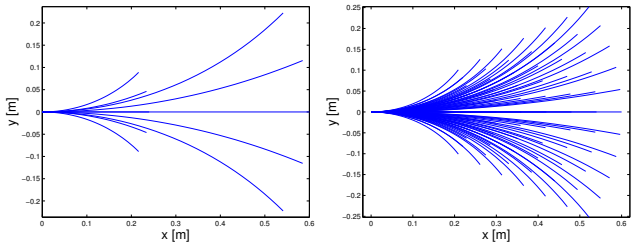


Figure 6: The motion primitive sets, $\mathcal{U}_{small}$ (left) with 10 motion primitives and $\mathcal{U}_{large}$ (right) with 77 motion primitives.

The POSQ parameters are $K_\rho = 1,\ K_\phi = -1,\ K_\alpha = 6,\ K_v = 3.8,\ \gamma = 0.15$. The two sets of motion primitives $\mathcal{U}_{small}$ with 10 controls and $\mathcal{U}_{large}$ with 77 controls are shown in Fig. 6. For the spline-based extend function we use η^3 splines (Piazzi, Bianco, and Romano 2007), seventh order polynomial spline whose paths have continuous tangent vectors, curvature and curvature derivatives along the arc length.

All extenders share the same integration time step t_s and velocity limits. For each combination of extender/map/planning algorithm we perform 100 runs and compute the average and standard deviation of all metrics. We use uniform sampling and the Euclidean distance as distance metric.

The RRT* cost function, derived from (LaValle and Kuffner 1999), has two terms, one for the approximated path length and one that measures heading changes along the path, both with equal weights ($w_d = w_q$)

$$C = \sum_{i=0}^{N_e-1} w_d ||\mathbf{P}_{i+1} - \mathbf{P}_i|| + w_q \left(1 - |\mathbf{q}_{i+1} \cdot \mathbf{q}_i|\right)^2 .$$

$N_e + 1$ are the intermediate points $\mathbf{P}_i$ of the local path and $\mathbf{q}_i$ the associated quaternions. The RRT* neighbor radius is constant at a high value with respect to the map size, we use a linear neighbor search.

Our implementation is based on the C++ SMP template library (Karaman). All experiments were running on an ordinary PC with 2.67 GHz Intel Core i7 and 10 GB of RAM.

4.1 Metrics

To quantify planning performance we compute the averages and standard deviations of the following metrics: tree size as the number of vertices (N_v), time to find a solution (RRT) or a first solution (RRT*) (T_s), and path length in meters (l_p).

Smoothness, although being an intuitive concept, is less straightforward to measure precisely. In (Balasubramanian, Melendez-Calderon, and Burdet 2012), Balasubramanian *et al.* survey a number of metrics to

quantify movement smoothness. We adopt the following measures that are relevant in our context.

Let v_{max} be the maximum magnitude of the robot velocity vector $\mathbf{v}$, $\tilde{\mathbf{v}} = \frac{\mathbf{v}(t)}{v_{max}}$ the normalized velocity, and $[t_1, t_2]$ the time interval over which the movement is performed.

1. η_{nmaJ}, the average of the mean absolute jerk normalized by v_{max}, for which the best value is zero:

$$\eta_{nmaJ} = -\frac{1}{v_{max}(t_2 - t_1)} \int_{t_1}^{t_2} \left| \frac{d^2\mathbf{v}}{dt^2} \right| dt,$$

2. average of the speed arc length η_{spal}, for which the best value is zero

$$\eta_{spal} = -ln \left(\int_{t_1}^{t_2} \sqrt{\left(\frac{1}{t_2 - t_1}\right)^2 + \left(\frac{d\tilde{\mathbf{v}}}{dt}\right)^2} dt \right),$$

3. average number of peaks η_{pm}

$$\eta_{pm} = -|\mathcal{V}_{peaks}|.$$

with $\mathcal{V}_{peaks} = \{\mathbf{v}(t) : \frac{d\mathbf{v}}{dt} = 0, \frac{d^2\mathbf{v}}{dt^2} < 0\}$ being the set of local velocity maxima.

4.2 Test Environments

Planning is carried out in three simulated environments (Fig. 7). In the *office environment*, there are few alternative ways to the goal. It has several local minima, the goal lies behind a U-shaped obstacle, and an asymmetry makes that the shortest path go through a narrow passage. The *hallway scenario* contains more open spaces and alternative paths to the goal. The *random map environment* contains 100 randomly placed square obstacles. There are many homotopy classes, some require more or less maneuvers than others. The map size in all scenarios is $50m \times 30m$.

5 Results and Discussion

The RRT results are given in Table 1, the RRT* results in Table 2. The best values in each metric are highlighted in bold.

With RRT as the planning algorithm, the proposed extend function *POSQ* outperforms motion primitives and splines in all metrics except path length. It produces smoother paths and finds the goal in less time with significantly smaller trees. The low number of tree vertices and the smaller planning times are mainly due to the ability of our approach to better follow the Voronoi bias and deeply enter unexplored regions of the configuration space. This is unlike, for example, motion primitives that require the concatenation of many small local expansions for the same exploration effort. In fact, all continuous extend functions that fully solve the two-point boundary value problem possess this property as also confirmed by the similar trends in the results of the spline-based extender.

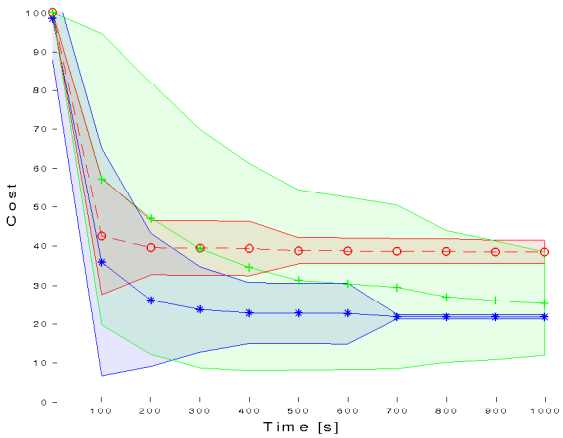


Figure 8: The RRT* cost C computed over 1000 seconds for the Random Map scenario. The trends are displayed (mean and standard deviation): in blue the POSQ results, in red the Motion Primitive ones and in green the splines. Our approach benefits most from the incremental character of RRT* and results in the lowest cost solution. The displayed trends are not changing with more planning time. See also Fig. 9.

The motion primitive extenders find shorter paths which is not surprising given the much denser trees from the multitude of small-sized extensions.

With RRT* as planner, our extender outperforms the other methods in tree size, path length and two of three smoothness measures. The fact that our method finds the shortest paths, and so the lowest cost in all the cases, suggests that it is particularly easy to rewire in the sense of the cost function, quite in contrast to motion primitives. This is also pointed out by Webb and van den Berg (Webb and van den Berg 2013) who state that the RRT* rewiring procedure is well suited for continuous extension approaches where reachability of a state is not compromised. Figure 8 is another indication in this direction. It shows an example cost trend when given more planning time. The POSQ extender can benefit most from the incremental path improvement of RRT*: in Fig.9 is showed a comparison between the paths obtained after 1000 seconds: definitely the POSQ generates the smoothest one.

While the proposed extender is smoothest in terms of the η_{spal} and η_{pm} measures, it falls behind the motion primitive approach in the jerk-related metric η_{nmaJ}. This may be explained by the much denser trees with several factors more vertices that allow solutions with fewer maneuvers. Regarding the time to find the first solution, T_s, the results are inconclusive. The high variance is mainly due to the large number of homotopy classes, particularly in the random map and hallway environments. POSQ and the spline-based extender (the two continuous approaches) grow the tree deeply into

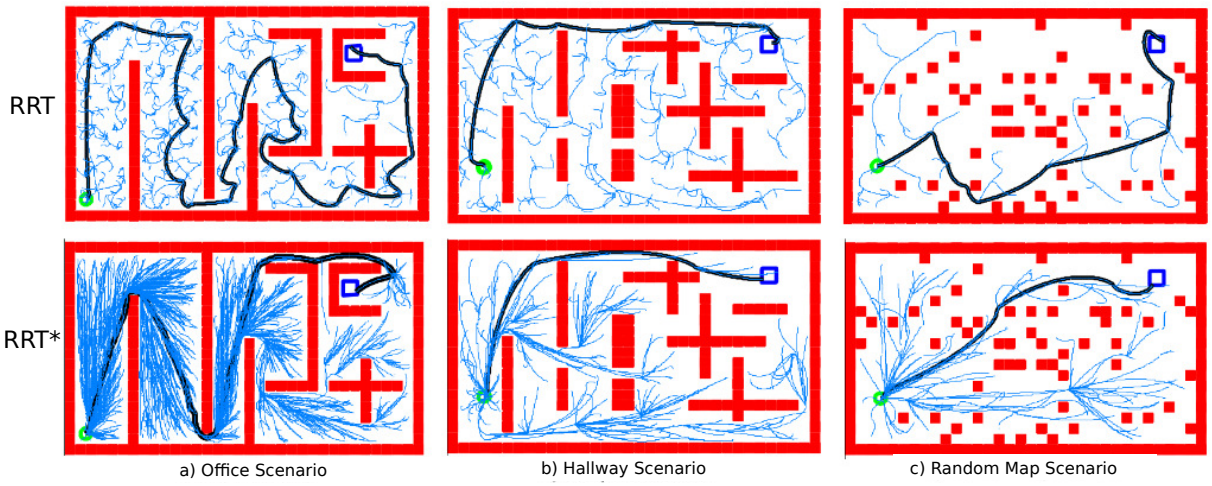


Figure 7: The three environments and example solutions found with the proposed extend function for RRT (top row) and RRT* (bottom row, showing the first solution). The start state (green circle) is always in the bottom left, the goal region (blue square) in the top right.

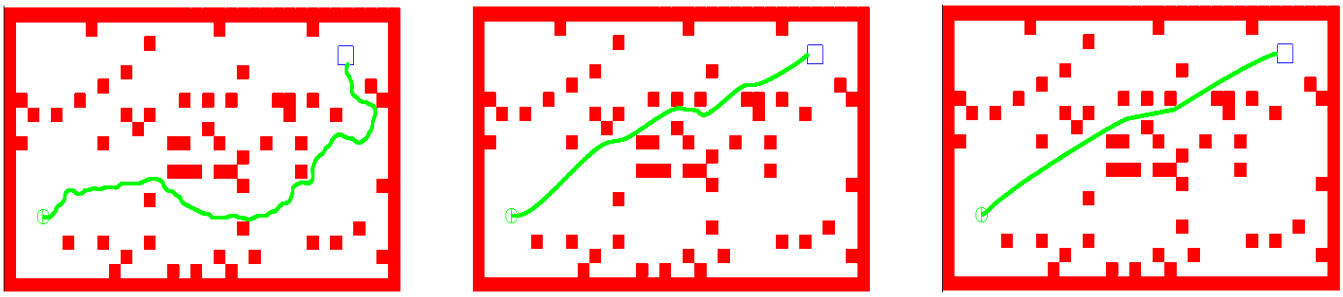


Figure 9: Typical RRT* solutions after 1000 seconds corresponding to the costs shown in Fig. 8. **Left:** the motion primitive approach generates the path with the high cost. **Center:** the spline-based extend function allows for a smoother path. **Right:** The path obtained by the POSQ extender is the smoothest one.

unexplored regions and discover many different ways to the goal, also inefficient ones at times. However, as discussed before, such first solutions can be improved when given more planning time, particularly well by the POSQ extender.

6 Conclusions

In this paper we have presented a novel RRT extend function for nonholonomic wheeled mobile robots. We have evaluated its impact on planning performance and path quality and found that it outperforms motion primitives and a spline-based approach in many relevant metrics. It enables RRT to find smoother paths in less time with smaller trees, and it enables RRT* to find shorter paths with smaller trees while being on par in planning time and smoothness. We also found that our method can benefit most from the cost-guided rewiring procedure of RRT* resulting in the lowest cost solutions when given more planning time.

In future work we will consider the extension of our method to kinodynamic models and the incorporation of recent RRT improvements such as regression avoidance and viability filtering. We also plan to develop specific heuristics for nonholonomic systems that help to select states in the tree for further expansion.

Acknowledgments

The authors thank Christoph Sprunk for valuable discussions and feedback. This work has partly been supported by the European Commission under contract number FP7-ICT-600877 (SPENCER)

References

Astolfi, A. 1999. Exponential stabilization of a wheeled mobile robot via discontinuous control. In *Journal of dynamic systems, measurement, and control*, volume 121.

Balasubramanian, S.; Melendez-Calderon, A.; and Burdet, E. 2012. A robust and sensitive metric for quantifying movement smoothness. *IEEE Transactions on Biomedical Engineering* 59(8).

Frazzoli, E.; Dahleh, M.; and Feron, E. 2005. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics* 21(6).

Office scenario						
Extenders	$\mathbf{N_v}$	$\mathbf{T_s}$ $[s]$	$\mathbf{l_p}$ $[m]$	$\eta_{\mathbf{nmaJ}}$	$\eta_{\mathbf{spal}}$	$\eta_{\mathbf{pm}}$
POSQ	**1667** ± 713	**0.197** ± 0.09	150.739 ± 18.446	$\mathbf{-0.051} \pm 0.006$	$\mathbf{-4.464} \pm 0.166$	**0** ± 0
MP $\mathcal{U}_{small}$	13335 ± 3283.4	2.235 ± 0.597	134.108 ± 8.687	-0.062 ± 0.0006	-5.8648 ± 0.064	37.8 ± 6.7
MP $\mathcal{U}_{large}$	14090 ± 3523.1	2.583 ± 0.720	**133.504** ± 8.85	-0.062 ± 0.0007	-5.8901 ± 0.066	21.09 ± 6.1
η^3 splines	2369 ± 939.9	0.274 ± 0.108	159.78 ± 14.88	-0.55 ± 0.06	-6.88 ± 0.16	**0** ± 0
Hallway scenario						
Extenders	$\mathbf{N_v}$	$\mathbf{T_s}$ $[s]$	$\mathbf{l_p}$ $[m]$	$\eta_{\mathbf{nmaJ}}$	$\eta_{\mathbf{spal}}$	$\eta_{\mathbf{pm}}$
POSQ	**520.4** ± 379.2	**0.050** ± 0.021	85.857 ± 16.740	$\mathbf{-0.039} \pm 0.007$	$\mathbf{-3.602} \pm 0.282$	**0** ± 0
MP $\mathcal{U}_{small}$	2458.3 ± 868.09	0.388 ± 0.124	72.918 ± 10.072	-0.0631 ± 0.001	-5.237 ± 0.138	22.32 ± 5.7
MP $\mathcal{U}_{large}$	2358.6 ± 922.2	0.367 ± 0.127	**71.734** ± 9.254	-0.0632 ± 0.001	-5.2528 ± 0.13	11.12 ± 4.4
η^3 splines	548.3 ± 514.5	0.0526 ± 0.026	86.659 ± 18.49	-0.382 ± 0.089	-5.93 ± 0.359	**0** ± 0
Random map scenario						
Extenders	$\mathbf{N_v}$	$\mathbf{T_s}$ $[s]$	$\mathbf{l_p}$ $[m]$	$\eta_{\mathbf{nmaJ}}$	$\eta_{\mathbf{spal}}$	$\eta_{\mathbf{pm}}$
POSQ	**277.2** ± 351.5	**0.031** ± 0.022	62.465 ± 9.003	$\mathbf{-0.027} \pm 0.007$	$\mathbf{-2.881} \pm 0.345$	**0** ± 0
MP $\mathcal{U}_{small}$	1095.1 ± 664.2	0.176 ± 0.104	**56.448** ± 5.242	-0.0638 ± 0.001	-4.977 ± 0.099	18.51 ± 4.8
MP $\mathcal{U}_{large}$	1124.6 ± 646.4	0.168 ± 0.09	57.27 ± 5.3	-0.0637 ± 0.001	-5.029 ± 0.098	9.48 ± 4.10
η^3 splines	519.6 ± 718.6	0.044 ± 0.035	66.686 ± 9.514	-0.3013 ± 0.082	-5.4851 ± 0.337	**0** ± 0

Table 1: RRT Results

Office scenario						
Extenders	$\mathbf{N_v}$	$\mathbf{T_s}$ $[s]$	$\mathbf{l_p}$ $[m]$	$\eta_{\mathbf{nmaJ}}$	$\eta_{\mathbf{spal}}$	$\eta_{\mathbf{pm}}$
POSQ	**1825** ± 785.7	$\mathbf{315.1} \pm 187.3$	**105.33** ± 4.96	-0.3261 ± 0.135	$\mathbf{-5.128} \pm 0.29$	**23.1** ± 11.1
MP $\mathcal{U}_{small}$	13571 ± 3601.8	731.3 ± 301.93	131.88 ± 8.35	$\mathbf{-0.0622} \pm 0.001$	-5.865 ± 0.06	38.59 ± 8.1
MP $\mathcal{U}_{large}$	14146 ± 2562.3	933.5 ± 258.50	134.257 ± 8.849	-0.0623 ± 0.001	-5.897 ± 0.07	30.65 ± 5.6
η^3 splines	3438 ± 4254.9	646.5 ± 483.62	116.4909 ± 6.337	-22.3 ± 6.81	-8.49 ± 0.09	47.30 ± 27
Hallway scenario						
Extenders	$\mathbf{N_v}$	$\mathbf{T_s}$ $[s]$	$\mathbf{l_p}$ $[m]$	$\eta_{\mathbf{nmaJ}}$	$\eta_{\mathbf{spal}}$	$\eta_{\mathbf{pm}}$
POSQ	**697.9** ± 704	66.9 ± 109.8	**54.16** ± 3.26	-0.1430 ± 0.126	$\mathbf{-3.6479} \pm 0.56$	**3.1** ± 4.26
MP $\mathcal{U}_{small}$	2385.3 ± 987	**51.3** ± 31.8	71.0350 ± 9.5819	-0.0631 ± 0.0007	-5.212 ± 0.129	16.2 ± 5.2
MP $\mathcal{U}_{large}$	2529.7 ± 1020.4	68.4711 ± 4.12	71.3390 ± 8.4327	$\mathbf{-0.0630} \pm 0.0002$	-5.2485 ± 0.12	11.3 ± 3.99
η^3 splines	3787.2 ± 14583	637 ± 2921	57.302 ± 4.2401	-9.365 ± 11.91	-7.08 ± 0.59	12.9 ± 13.9
Random map scenario						
Extenders	$\mathbf{N_v}$	$\mathbf{T_s}$ $[s]$	$\mathbf{l_p}$ $[m]$	$\eta_{\mathbf{nmaJ}}$	$\eta_{\mathbf{spal}}$	$\eta_{\mathbf{pm}}$
POSQ	**400.8** ± 551.2	43.16 ± 90.54	**46.11** ± 2.4	-0.0896 ± 0.103	$\mathbf{-2.95} \pm 0.697$	**1.3** ± 2.71
MP $\mathcal{U}_{small}$	1028.1 ± 597	$\mathbf{13.11} \pm 12.48$	56.66 ± 5.08	$\mathbf{-0.0636} \pm 0.0007$	-4.98 ± 0.10	18.7 ± 4.95
MP $\mathcal{U}_{large}$	998.7 ± 535.8	15.4411 ± 14	56.9105 ± 4.7867	-0.0637 ± 0.0005	-5.02 ± 0.08	10.09 ± 4.2
η^3 splines	815.7 ± 2421.1	157.8 ± 715.8	48.5212 ± 2.7370	-7.67 ± 11.87	-6.46 ± 0.97	5.0 ± 7.54

Table 2: RRT* Results

Goretkin, G.; Perez, A.; Platt, R.; and Konidaris, G. 2013. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *Int. Conf. on Robotics and Automation (ICRA)*, 2429–2436.

Hwan Jeon, J.; Karaman, S.; and Frazzoli, E. 2011. Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*. In *Decision and Control and European Control Conference (CDC-ECC)*.

Kalisiak, M., and van de Panne, M. 2006. RRT-blossom: RRT with a local flood-fill behavior. In *Int. Conf. on Robotics and Automation (ICRA)*.

Kalisiak, M., and van de Panne, M. 2007. Faster motion planning using learned local viability models. In *Int. Conf. on Robotics and Automation (ICRA)*.

Karaman, S., and Frazzoli, E. 2010. Incremental sampling-based algorithms for optimal motion planning. In *Proc. of Robotics: Science and Systems (RSS)*.

Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. In *Int. Journal of Robotics Research (IJRR)*, volume 30.

Karaman, S., and Frazzoli, E. 2013. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 5041–5047. IEEE.

Karaman, S. Sampling-based Motion Planning (SMP) Template Library. `http://www.mit.edu/~sertac/smp_doc`.

Kuwata, Y.; Karaman, S.; Teo, J.; Frazzoli, E.; How, J.; and Fiore, G. 2009. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology* 17(5).

Laumond, J.-P.; Sekhavat, S.; and Lamiraux, F. 1998. *Guidelines in nonholonomic motion planning for mobile robots*. Springer.

LaValle, S., and Kuffner, J. 1999. Randomized kinodynamic planning. In *Int. Conf. on Robotics and Automation (ICRA)*.

Perez, A.; Platt, R.; Konidaris, G.; Kaelbling, L.; and

Lozano-Perez, T. 2012. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *Int. Conf. on Robotics and Automation (ICRA)*.

Piazzi, A.; Bianco, C.; and Romano, M. 2007. η^3-splines for the smooth path generation of wheeled mobile robots. *IEEE Transactions on Robotics* 23(5).

Webb, D., and van den Berg, J. 2013. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *Int. Conf. on Robotics and Automation (ICRA)*.

Yang, K.; Moon, S.; Yoo, S.; Kang, J.; Doh, N.; Kim, H.; and Joo, S. 2014. Spline-based rrt path planner for non-holonomic robots. *Journal of Intelligent and Robotic Systems* 73(1-4).

Performance Level Profiles

Ronen I. Brafman
Computer Science Department
Ben-Gurion University
brafman@cs.bgu.ac.il

Guy Shani
Information Systems Engineering Department
Ben-Gurion University
shanigu@bgu.ac.il

Solomon Eyal Shimony
Computer Science Department
Ben-Gurion University
shimony@bgu.ac.il

Abstract

We define *performance-level profiles (PLPs)* which can serve as a basis for performance level agreements (PLAs) between robot or robotic-module designers and their customers/users. PLPs provide a formal description of the commitments of a module. They describe the conditions under which the module should be operated (i.e., what conditions must hold in the environment and the system, what other modules may or may not operate concurrently, and what resources are required), and the expected result of its operation under these conditions. PLPs are motivated and are closely related to action specification languages, such as PDDL 2.1. Yet, they differ from these languages, introducing concepts that more naturally fit robotic modules. PLPs have a number of possible roles: First, as a formal specification language that is much more precise than less formal specification documents, such as SSS. Second, as a language for providing guarantees to module users/customers, in the form of a PLP-based *performance-level agreement (PLA)*. Third, as input to automated reasoning tools providing support for module monitoring online, for validating proposed execution plans, for selecting among alternative execution plans, and, eventually, to automatically generate execution plans. We describe PLPs and explain the type of infrastructure support we provide for them in ROS.

Introduction

Web service and network infrastructure providers often sign service-level agreements (SLAs) with their users. These agreements describe *minimal* commitments regarding the functionality of these services that the user can rely upon, providing the latter with some confidence and guidelines regarding what it can expect from the service. SLAs are not precise specifications of the service, but provide a good-enough abstraction for most users. We would like to suggest that a similar concept is useful and appropriate in the context of robotics, and especially autonomous robots.

Indeed, if robotic infrastructure is to reach a plug-and-play state, as in personal computing, we must be able to combine diverse robotic modules and components easily. Yet, as noted by Abdellatif et al. (2012): "Systems built by assembling together independently developed and delivered components often exhibit pathological behavior. Part of the problem is that developers of these systems do not have a precise way of expressing the behavior of components at their interfaces, where inconsistencies may occur."

Although it is likely to be very difficult to provide a precise specification of the behavior of a module given an open-ended world, we believe that it is reasonable to provide a specification that circumscribes the conditions under which the module is likely to execute as expected, and some minimal probability of successful execution given that these conditions hold. Such a specification, which we call a *performance-level profile (PLP)* can form the basis of performance level commitments by module designers, forming what can be called *performance-level agreement (PLA)*. That is, a PLA is an agreement/commitment/contract in the form of a PLP.

PLPs, introduced and described in this paper, are motivated by the above considerations, that closely resonate with issues we encountered during a joint project with industrial (Israel Air Industries and Cogniteam) and academic partners. First, our industrial partners complained about the lack of ability to provide customers with some reasonable performance guarantees. Second, it was evident that in a collaborative project, with different modules supplied by different partners, it will be difficult to validate various sequences of operations, or to generate them automatically, without a suitable, formal specification of the functional modules. Third, although formal methods for development of modules with guarantees and clean specifications are available (e.g.,BIP (Basu, Bozga, and Sifakis, 2006)), it was unrealistic to expect these practitioners to change their software development practices and software development tools for this project. PLPs were introduced to address these issues.

PLPs provide a formal (possibly partial) description of functional modules. They are agnostic to the method using which the module was built, and borrow many ideas from action description languages such as PDDL 2.1 (Fox and Long, 2003), probabilistic PDDL (Younes and Littman, 2004), and RDDL (Sanner, 2010). They also integrate the well known ideas of achievement and maintenance goals (Kaminka et al., 2007), as well as a new *repeat* construct which allows us to make explicit the frequency by which input parameters are read or provided and output parameters are published, which are often quite important in the design of robotic modules and influence various error parameters.

PLAs/PLPs have a number of potentially important roles.

First, they provide a more formal methodology that can replace the much used SSS (systems/subsystem requirements specification). SSS are often used by system engineers to describe requirements from a software module. SSS have a rigid structure, but the content of their fields is textual. PLPs can provide for a more formal language with better support in code – as we shall see later. Second, as noted above, they can be used as a basis for agreements/commitments made by module designers to module users. Third, as formal commitments, they can be used as input to automated reasoning tools that can use them for monitoring, validation, and ultimately, automated planning. In particular, we are working on simple tools that will make it very easy to use them as input to a third-party monitoring tool available for ROS. These tools make it easy to specify, initially, PLPs with abstract conditions, and to eventually turn them into conditions grounded in code.

PLPs: An Overview

The main objective of a PLP is to clarify the end-effect of executing the module. As a simple example, imagine a module designed to grasp an object. The expected outcome is that the object is held by the arm. However, this effect is not guaranteed (in most realistic settings). There is some probability of failure, and failure can come with some side effects, such as the object falling, or being broken. Moreover, the probability of success and failure may depend on various properties, such as the shape and size of the object. Furthermore, it may be unrealistic to specify the success probability in general, as there are too many external things that could impact it. For example, if another arm is attempting to catch the object at the same time, it is difficult to predict which one will succeed.

PLPs have two abstract components. The first circumscribes the conditions under which the profile is provided – conditions that must hold for it to be valid. Such conditions include things that must hold when it is performed and the resources it requires. The second specifies the effect of the action – what success means, what are possible failure modes, and what is the probability of each.

There are four types of PLP, corresponding to four types of modules. *Achieve* modules attempt to achieve a new state of the world. For example, changing the orientation of the robot to some goal orientation. *Maintain* modules attempt to maintain some property. For example, a module that maintains some orientation; or, a module that ensures that the robot remains within some confined area. We allow for maintain modules that maintain a property that is not necessarily true initially. For example, "maintain speed of 10km/hr". Thus, essentially, these modules need to make the condition true and maintain it. In principle, one could separate this into two phases: achieve, then maintain; but this separation is not natural in many cases, especially when the module is essentially a closed-loop controller that continuously attempts to reduce the difference between the current condition and the target condition. The third module type is an *Observe* module – a module that attempts to recognize some property of the current state of the world. For example, the robot's location, or whether there is a cup on the table. Finally, the fourth type is *Detect* which can be viewed as the *maintain* version of *observe*.

Many robotic modules operate by repeatedly updating or modifying some data-structure or signal based on information that is constantly being updated. Here, important issues are the frequency by which input and output are updated. To model such constructs we introduce the novel *repeat* wrapper for the *achieve* and *observe* modules. For example, a mapping module constantly updates a map of the environment as it obtains information from relevant sensors. Concurrently, a path-planning module may be updating the path as it obtains new maps. Each such module can be viewed as performing an *achieve* task repeatedly.

One may argue that with the introduction of *repeat* wrapper, *maintain* and *detect* is redundant, as *maintain* can be implemented as repeat-achieve, while *detect* can be implemented as repeat-observe. Nevertheless, we decided to keep these constructs around for user convenience.

PLPs: Technical Specification

Variables and Resources

The formal definition of PLPs rests on the specification of properties of states of the world. These are defined by specifying properties of various state variables. It is desirable to have a coherent specification of such variables, so that the relationship between modules is clear. The meaning of these variables – their relationship to the real world, i.e., the correspondence between the model and the real world must be clear for the specification to be meaningful. Some state variables can refer to local system parameters, of course.

In addition, each module may need access to certain resources. These resources could be energy or memory, or they could be some actuator, or some region of space. These must be specified, much like state variables, and coherent and consistent use of these names is required. In fact, resources can be viewed as a special class of state variables, whose state indicates the status of the resource (e.g., available, > 100 gallons, etc.). However, because we believe that they carry special significance to programmers and operators, we distinguish them from other variables.

Software support for consistent use and maintenance of the list of variables and resources is advisable and is part of the set of tools supporting PLPs that we are constructing.

Common Elements

All modules specify the following elements:

Parameters: Variables supplied to the module as input or provided by the module as its output. Some input parameters can also be output parameters (i.e., after undergoing some processing). We point out that one class of parameters could be *error* parameters. That is, parameters that specify the accuracy of outer parameters. For example, localization is often performed by using a filter, from which an error estimate can be obtained (e.g., the covariance matrix in a Kalman Filter).

Set of variables: Local variables and their range.

Application Context: A set of conditions, described below, specifying the context under which the PLP is valid.

The application context contains the following elements.

Required resources: List of resources required. If the resource is quantifiable, a required quantity is mentioned. If the resource is needed for operation and then freed (e.g., memory, some tool, some actuator), the requirement status must be mentioned. Possible values are "exclusive" or some frequency of use (although frequency is more naturally captured using the *repeat* wrapper.

(optional) Maximal rate of change: Maximal change in resource level per time unit.

Preconditions: conditions on the world at the start of execution time under which the PLP is defined. These conditions can refer only to parameters.

Concurrency conditions: conditions on the world during execution under which the PLP is defined. These can be on parameters as well as local variables.

Concurrent modules: conditions on modules that must or must-not be executed concurrently. These fields specify the conditions under which the PLP is defined. That is, a PLA based on this PLP makes no guarantees if these conditions are violated.

(optional) Parameter Frequency: The frequency by which each parameter must be read. For example, localization information may be required throughout the execution of various navigation tasks. The frequency by which a parameter is updated together with its accuracy (which is available if an error parameter exists) can affect the accuracy of output. For example, a navigation module that aims to reach a specified position may need to obtain position information with certain rate and certain accuracy to ensure success.

Semantically, *required resources* as well as *maximal rate of change* are special types of concurrency conditions that apply to availability and usage of resources. They do not describe resource consumption, though, which one should specify in the *side-effects* field.

Side-effects: Each module has an intended effect – module types differ in the nature of this intended effect. However, each module may also have side-effects that are a result of executing this module, but are not a measure of its success or failure. For example, if the module consumes some resource, a natural side effect is that the level of this resource has declined. Side effects are described by a conditional assignment to a parameter, which could depend on a local variable (such as running time, or distance traveled). Intuitively, side-effects are changes caused by the module that could potentially impact other modules.

Achieve Modules

Achievement modules attempt to change the state of the world so that some desirable property will hold. For example, fuel tank is full, robot is standing, plane has landed, etc. In addition to the common elements, their PLP contains the following:

Achievement goal: A condition defined on the parameters that is achieved by the action.

(optional) Failure modes: Possible ways in which the module could fail to achieve the goal. That is, these are conditions that are inconsistent with the achievement goal that could be the outcome of the action.

Probability of success. It may be conditional on properties of the world, and expressed via parameters.

Failure probability: For each failure mode, its probability, possibly conditional on some parameter values).

Running time distribution given success. Possibly conditional on various parameters, e.g., Rayleigh(f(path-length)]

Running time distribution given failure. Possibly conditional on various parameters, e.g., Rayleigh(g(path-length)]

Example: achieve "holding(cup)"

- Set of parameters: arm-empty, wet(cup), clear-path(cup)
- Achievement goal: holding(cup)
- Requires resources: robot-arm (exclusive)
- Preconditions: arm-empty, clear-path(cup), cup-ok
- Concurrency conditions: none
- Concurrent modules: none
- Side effects: ¬arm-empty
- Failure modes: (1) ¬holding(cup) (2) ¬holding(cup) and broken(cup)
- Probability of success: 0.9 if ¬wet(cup), 0.3 if wet(cup)
- Failure probability: (1) 0.08 if ¬wet(cup), 0.4 if wet(cup) (2) 0.02 if ¬wet(cup), 0.3 if wet(cup)
- Running time given success: Normal(60,20).
- Running time given failure: Normal(80,40).

Maintain Modules

Maintain modules attempt to maintain the value of some variable or the truth value of some more complex condition, such as maintain heading, maintain speed, maintain perimeter clean, etc. In addition to the common elements, their PLP contains the following:

Maintained condition Defined over parameters.

Initially true? A boolean value that indicates if the module expects the condition to be true initially.[1]

Success Termination condition: Condition under which module stops operating (could be time). This is a condition that does not violate the maintenance goal.

Failure Termination conditions: Conditions under which module stops operating (could be time). These are conditions that indicates some type of failure. There may be multiple such conditions.

[1]Formally, this is just another precondition. But because of its central role, it is designated as a special field.

(optional) failure modes: Possible ways in which the module could fail to maintain the condition.

Probability of success. Possibly conditional on properties of the world.

Failure probability. For each mode, probability may be conditional on properties of the world.

Running time distribution given success, and possibly other conditions of the world.

(optional) Running time distribution given failure, and possibly other conditions of the world.

Sometimes, one needs to maintain some condition in order to achieve a goal. For example, one may reach a target position by maintaining a pre-computed path to the goal, either by iteratively reaching way-points, or by ensure that the heading is always in the direction of the path. One can model this as either an achieve module, whose goal is to reach the target position, or as a maintain module that maintains heading along the path direction, with the success termination condition being "at-the-goal". The advantage of the latter definition that it less abstract, specifying that the goal is reached by following a certain path. Thus, it is easier to detect problems by monitoring the path and alerting the operator or system whenever that actual heading is not in the direction of the path.

Example: Maintain direction based on map until goal is reached.

- Parameters: Map, goal location.
- Variables: none.
- Maintained condition: vehicle points in direction of path at its current location.
- Success Termination condition: Within K meters from goal.
- Failure Termination condition: position unknown or too far from path
- Side effects: Gas consumption.
- (optional) failure modes: (1) vehicle off-road, undamaged, (2) vehicle off-road, damaged
- Probability of success: Given dry: 0.95 wet: 0.8
- Failure probability: Given dry: (1) vehicle off-road, undamaged 0.045 (2) vehicle off-road, damaged 0.005. Given wet: (1) vehicle off-road, undamaged 0.15 (2) vehicle off-road, damaged 0.05
- Running time given success: f(path-length).
- Required resources: fuel, steering wheel (exclusive)
- Preconditions: Path tangent rate of change$< c$
- Concurrency conditions: no other car on path

Observe Modules

Observe modules attempt to identify the value of some variable(s) in the current world state. For example, observe distance to wall, observe whether robot is standing, observe whether object is held, etc. In addition to the common elements, their PLP contains the following:

Observation goal: the boolean condition (defined over parameters) or parameter whose value is observed.

Probability of failure to observe

(boolean) Probability that observation is correct. Quantifies this probability under the assumption that the observation was successfully performed. This value can be conditional on various properties of the world expressed via parameter values.

(continuous) One of:

- Pr(real value | observed value). This distribution is based on the assumption that the observation was successfully performed. It relates the real value of the parameter observed with the observed value. It can depend conditionally on properties of the world.
- Confidence interval + confidence level.

Running time distribution given success. May depend on additional parameters.

Running time distribution given failure. May depend on additional parameters.

Example: Observe "wall-ahead"

- Set of variables: $\emptyset$
- Parameters: laser scanner output
- Observation goal: Distance to wall ahead$< c$
- Requires resources: Laser scanner
- Preconditions: laser ok
- Concurrency conditions: $\emptyset$
- Concurrent modules:$\emptyset$
- Side effects: $\emptyset$
- Failure modes: sensor malfunction
- Probability of failure to observe: 0.001
- Probability observation is correct: 0.95
- Running time distribution given success: 1 millisecond
- Running time distribution given failure: 1 millisecond

Detect Modules

Detect modules are related to observe modules much as maintain modules are related to achieve modules. Their goal is to detect some change. Thus, they are not intended to identify the current state of the world, but rather, to observe it over time until some condition holds. Thus, they can be implemented using a maintain module that repeatedly observes. For example, detect intruder, detect temperature change, detect motion, etc. In addition to the common elements, their PLP contains the following:

Detection goal: the condition that is being detected (over parameters).

Probability of successful detection given condition (possibly conditional on properties of the world).

Example: Detect wall

Set of variables: none

Parameters: Laser scanner output

Frequency: 30hz

Detection goal: Distance to wall in range [0.5,1] meters.

Side effects: none

Probability of successful detection: 0.95

Requires resources: Laser scanner

Preconditions: Distance to wall $>$ 0.5 meter.

Concurrency conditions: none

Concurrent modules: Laser scanner

Repeat

In many robotic applications, various modules run continuously, updating some data-structure, such as a map, or a path, or monitoring the environment for some trigger, such as detection of an intruder. Such modules are essentially loops that execute some underlying routine many times, e.g., map and path update, or repeatedly analyze some input until a condition holds. While in code terms, they offer nothing special, in terms of their spec, they raise a number of issues – for example, the rate in which the update occurs, the rate in which input is expected to be received, and the termination condition. For this purpose we provide a special boolean *repeat* field for each PLP. When its value is true, additional information must be supplied including:

Execution Frequency – How many times per second is the underlying module executed. We assume that if the module has an output parameter, then the frequency it is updated is the same as the execution frequency.

Input Frequency – For each input parameter, it is possible (optional) to specify its expected update frequency. One can consider this as a special type of concurrency condition. If some parameters are really global variables (such as fuel-level), they may be obtained on demand (i.e., by calls within the code).

Termination Condition – Once this condition is true, the module stops executing the loop.

An interesting issue regarding repeat modules is the automated derivation of the parameters of the entire module given the properties of the repeat construct (frequencies, termination condition) and the properties of the module that is repeatedly executed (i.e., success probability, running time). For example, suppose that one has computed a map of the environment, with some accuracy, and has now generated a path based on this map. At some positions along this path the distance to the nearest obstacle is 50cm. This implies that the localization error must be smaller. This, in turn, likely requires more accurate localization, that can potentially be obtained by increasing the rate by which images, or scanner readings are obtained and analyzed. We believe that automatic inference of such constraints could be valuable in many applications, and we pose this problem as an important question for future work.

PLP Tools and ROS

We developed a number of tools for specifying and using PLPs.

PLP Editor

The simplest tool is a PLP editor. The editor provides simple, GUI-based support for PLP generation by, essentially filling in a form using a convenient GUI. Once a PLP is generated using the editor, the editor is able to generate three types of output:

1. Text description.
2. Computer-readable PLP in XML format.
3. Code template – a piece of code, with some possibly missing definitions that a programmer can insert into her code. Note that this code is not executable, it is simply an encoding of the PLP with suitable keywords that can be read by a pre-processor, as explained next.

Code Support – ROS Integration

Programmers will be able to introduce PLP descriptions into their code using appropriate keywords. A preprocessor will parse this input – recognizing the key words – and will generate into an associated library a description of the PLP in XML format. The benefit of this method is that the programmer can use – in fact, should use – only variables defined in the code, and can write explicit code using the programming language used to define PLP preconditions, etc.

As noted above, the PLP Editor will be able to generate code fragments, possibly leaving to the programmer the specification of some conditions in code as well as providing for each parameter a link to the relevant ROS topics from which it can be obtained. Thus, an initial spec can be provided by a system engineer using the PLP, imported into the code, and then farther instantiated by the programmer. This provides very convenient support for top-down generation of PLPs. First – describe the PLP abstractly using human-readable conditions, then instantiate these conditions in code, leaving the abstract description as an annotation.

In principle, the programmer can define PLPs for various functions in her code – whether visible externally or not. There are certain benefits to this, since given such definitions, she is able to utilize any tool that uses PLPs to automate various tasks. Each such PLP will come with a unique identifier that will allow the association of the PLP with the relevant code. Ideally, we would like each service provided by a ROS node to come with its associated PLPs.

Integrated Monitoring Support

As part of its support for decision-making algorithms within ROS, Cogniteam `www.cogniteam.com` has recently provided a monitoring module `wiki.ros.org/scriptable_monitoring`. This module runs in parallel with other modules, and monitors their execution. The monitor is simply a piece of code that evaluates conditions it is fed, and issues alerts when these conditions are violated or become true, as required.

PLPs defined in code enjoy automated integration with this module. Specifically, the preconditions, concurrency conditions, and effects of a module are processed into conditions that are fed into the monitoring module. These conditions correspond to the preconditions, concurrency conditions, termination conditions, and expected outcomes of the modules. In addition, condition that monitor execution time are generated, and alert the operator in case the module executes too long. For example, if the module has not stopped after a time that corresponds to the average running time + two standard-deviations. These alert help operators (or automated decision making modules) make informed choices and recover from errors.

While this is simple from an implementation perspective, the integrated and automated handling of these condition checks provides much convenience to the programmer as well as important run-time information to the operator.

Play-outs and Planning

PLPs can be used for predicting the behavior of modules, and in particular, the behavior of module combinations, such as modules that run in sequence or in parallel, or in a loop. This calls for a theory of PLP composition, which is the subject of ongoing work. In difference to earlier work such as Lesire, Doose, and Cassé (2011) which attempts to predict worst-case behavior, we are attempting to estimate distributions over running times, or their moments. The capability to play-out can be used by operators or automated tools in order to assess the suitability of different macros for various tasks online or offline. In addition, this could be the input to future planning algorithms.

Related Work

PLPs for achievement goals are based on existing action languages, mixing features from a number of sources. The idea of preconditions and effects goes back to STRIPS (Fikes and Nilsson, 1971). The use of concurrency conditions was integrated into PDDL when it was extended to handle temporal actions (Fox and Long, 2003), which obviously included a specification of the running time of operators, as well as the ability to specify resources and their consumption over time. PLPs emphasize a slightly different version of concurrency condition, which attempts to address the issue of action (here, module) interaction introduced by Boutilier and Brafman (2001). The idea of requiring exclusive use of a resource is also a technique for preventing harmful interactions, or simply interactions whose effect cannot be predicted. Thus, a module can require exclusive use of an arm, preventing other modules from interfering with its use. This idea goes back to the classical use of mutex in concurrent systems (Dijkstra, 1965), and is implemented in some languages for concurrent programming. A related idea appears in Structured Reactive Controllers (Beetz, 1999) where the notion of embedability is defined. That notion asserts that a certain module may be executed concurrently with a set of other modules without their execution interfering with its ability to reach its goal.

In addition, PLPs address uncertainty by borrowing ideas from PPDDL (Younes and Littman, 2004) and RDDL (Sanner, 2010). For each of these aspects (time, concurrency, uncertainty, resources) there are languages that provide more powerful constructs, whereas PLPs attempt to address all essential aspects of the performance of a module, while providing a good tradeoff between expressivity and intuitiveness. Thus, while PDDL2.1 can describe temporal actions, it does not describe actions with stochastic durations, and while RDDL describes probabilistic effects, it does not specify temporally extended actions, etc.

Maintenance goals are a well know concept, but we have not seen them come up in planning operator languages before. In principle, a maintenance goal could be specified using a version of temporally extended actions. In one version of temporally extended actions discussed by Fox and Long (2003), one may specify effects that take effect immediately at the beginning and throughout the execution of the action. Such specification is a bit counterintuitive, though, as maintain modules usually maintain a condition that is already true. Thus, the action would have the condition to be maintained both as a precondition and an effect. *detect* and *observe* modules, on the other hand, are motivated by the ideas of observation in POMDPs and contingent planning, whereas the ideas behind the *repeat* module, and in particular, the introduction of input/output frequencies appears new, to the best of our knowledge.

Overall, with the exception of *repeat*, none of the constructs underlying PLPs is new, but their combination covers essential aspects of robotics modules, and includes components essential to the proper operation and use of functional modules, while attempting to remains intuitive.

The approach taken by PLPs is minimalist in the sense that it does not dictate (nor provide) particular architecture or tools for module construction. There is no doubt that tools such as BIP (Basu, Bozga, and Sifakis, 2006; Basu et al., 2008) which provide a methodology for top-down construction of modules as well as tools for validating properties of the constructed system, or techniques for automated controller generation such as Kress-Gazit and Pappas (2010) provide more powerful support for the synthesis and construction of systems with guarantees. Nevertheless, little code is generated today using formal specification methods, and we expect that this state of affairs will be true in the area of robot design. On the other hand, software engineering methods are commonly used in the design of large software systems. PLPs and their ability to define PLAs provide a specification method that is similar in some ways to the commonly used SSS, yet more formal, and fits well within a methodology in which a systems engineer provides an abstract, almost textual specification, that is then implemented in code by a programmer. They also provide a way for a designer to explain/guarantee the functional behavior of its module at some level.

Summary and Future Work

We described a language for specifying the properties of software modules, motivated by perceived needs of robotics application. Our original motivation stems from discussions with our industry partners who lack a language and

a methodology for providing their clients with reasonable performance guarantees for their products. This has led us to suggest an idea similar to service-level agreements, common in telecommunications industry, and more recently in the area of web services. Robots, and especially autonomous robots operate in a much more open-ended environment, requiring more complicated notions. While our language is derived from standard planning languages, it attempts to make the specification more intuitive and appropriate by resorting to the well known notions of achievement and maintenance goals, adding also analogous notions for the sensing-side of robot activity. In addition, using the *repeat* wrapper, we are able to naturally give a clear role to the notion of frequency and latency, so commonly used by designers of robotic hardware and software.

Our current experience shows that a specification of an initial PLP for a module is relatively easy. However, the PLP will likely miss out on some information. However, with the aid of the editor embedded code feature, an iterative process is easy to support.

The value of PLP use will depend on future developments. However, our integration of PLPs with Cogniteam's monitoring tools already provides some value even when the specification is partial. It provides operators with useful information about unexpected or problematic conditions, such as modules operating without the required preconditions and concurrent conditions, information about running times, and effects of modules used online. Together with automated reasoning tool – both for playing out various scenarios and for planning, we believe that PLPs can form an important tool for a more convenient, powerful, and formal methodology for the design and control of autonomous agents.

To realize these possible benefits, the PLP definition may require farther enhancement to cover additional aspects of module performance. For example, we are now considering issues such as partial success – e.g., sometimes a module that is unable to achieve its desired goal, e.g., because of some condition that is violated, may still be able to achieve an alternative one, or perhaps ensure some safe end-state. Similarly, often modules are designed with a back-up module that is called in the case of failure. While each such module can be specified independently, it may be desirable to have a special structure for module/backup pairs. Another possible enhancement is a more refined process model. Some modules, provide more complex services that require modeling some notion of state. Indeed, many of the formal tools developed for principled construction of robotic modules supply such a model, e.g., a state-machine (Basu et al., 2008) or a Petri-Net (Montano, Garcia, and Villarroel, 2000). We wish PLPs to remain abstract specification and not nearly executable process models. Yet, the exact level of abstraction that will provide sufficient functionality remains an open question for future research.

Acknowledgements The authors would like to thank the anonymous reviewers for their useful comments and useful pointers. Brafman and Shani are supported in part by ISF Grant 933/13. Brafman and Shimony are supported in part by the Lynn and William Frankel Center for Computer Science.

References

Abdellatif, T.; Bensalem, S.; Combaz, J.; de Silva, L.; and Ingrand, F. 2012. Rigorous design of robot software. *Robotics and Autonomous Systems* 60(12):1563–1578.

Basu, A.; Gallien, M.; Lesire, C.; Nguyen, T.-H.; Bensalem, S.; Ingrand, F.; and Sifakis, J. 2008. Incremental component-based construction and verification of a robotic system. In *European Conference on AI*.

Basu, A.; Bozga, M.; and Sifakis, J. 2006. Modeling heterogeneous real-time components in bip. In *International Conference on Software Engineering and Formal Methods (SEFM-06)*, 3–12.

Beetz, M. 1999. Structured reactive controllers: Controlling robots that perform everyday activity. In *Agents*, 228–235.

Boutilier, C., and Brafman, R. I. 2001. Planing with concurrent interacting actions. *Journal of AI Research* 14:105–136.

Dijkstra, E. W. 1965. Solution of a problem in concurrent programming control. *Communication of the ACM* 8(9):569.

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Aritificial Intelligence* 2:189–208.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of AI Research* 20:61–124.

Kaminka, G. A.; Yakir, A.; Erusalimchik, D.; and Cohen-Nov, N. 2007. Towards collaborative task and team maintenance. In *Autonomous Agents and Multi-Agent Systems*.

Kress-Gazit, H., and Pappas, G. J. 2010. Automatic synthesis of robot controllers for tasks with locative prepositions. In *ICRA*, 3215–3220.

Lesire, C.; Doose, D.; and Cassé, H. 2011. Validation of real-time properties of a robotic software architecture. In *6th National Conference on Control Architectures for Robots*.

Montano, L.; Garcia, F.; and Villarroel, J. 2000. Using the time petri net formalism for specification, validation, and code generation in robot-control applications. *International Journal of Robotics Research (IJRR)* 19(1):59–76.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description.

Younes, H., and Littman, M. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

Iterative Goal Refinement for Robotics

Mark Roberts[1], Swaroop Vattam[1], Ronald Alford[2], Bryan Auslander[3], Justin Karneeb[3], Matthew Molineaux[3], Tom Apker[4], Mark Wilson[5], James McMahon[6], and David W. Aha[5]

[1]NRC Postdoctoral Fellow; Naval Research Laboratory, Code 5514; Washington, DC
[2]ASEE Postdoctoral Fellow; Naval Research Laboratory, Code 5514; Washington, DC
[3]Knexus Research Corporation; Springfield, VA
[4]Exelis Corporation; Alexandria, VA
[5]Navy Center for Applied Research in Artificial Intelligence; Naval Research Laboratory, Code 5514; Washington, DC
[6]Physical Acoustics Branch; Naval Research Laboratory, Code 7130; Washington, DC
[1,2]*first.last*.ctr@nrl.navy.mil | [3]*first.last*@knexusresearch.com | [4]thomas.apker@exelisinc.com | [5,6]*first.last*@nrl.navy.mil

Abstract

Goal Reasoning (GR) concerns actors that assume the responsibility for dynamically selecting the goals they pursue. Our focus is on modelling an actor's decision making when they encounter notable events. We model GR as an iterative refinement process, where constraints introduced for each abstraction layer shape the solutions for successive layers. Our model provides a conceptual framework for robotics researchers and practitioners. We present a goal lifecycle and define a formal model for GR that (1) relates distinct disciplines concerning actors that operate on goals, and (2) provides a way to evaluate actors. We introduce GR using an example on waypoint navigation and outline its application, in three projects, for controlling simulated and real-world vehicles. We emphasize the relation of GR to planning, and encourage PlanRob researchers to collaborate in exploring this exciting frontier.

1. Introduction

Robotic systems often act using incomplete models in environments that are dynamic, partially observable, and non-deterministic. One consequence is that they will encounter notable events. Appropriate responses to notable events can be *designed* a priori or *learned* by the actor. During execution, robots *deliberate* on their responses to notable events and can choose to adjust their expectations or world model, repair their current plan, replan, or regoal (i.e., change their current goal(s)). In each case, they take steps toward achieving goals. We refer to this capability of reasoning about ones goals as *goal reasoning* (GR), which involves dynamically assessing the tradeoffs within the space of goals. We argue that GR is of particular value to robotic systems, as it supports more autonomous behavior.

We present a preliminary formal model that frames GR as an iterative refinement process similar to planning as refinement search (Kambhampati *et al.*, 1995) and iterative repair (Chien *et al.*, 2000). Our model provides a common language for defining GR actors and complements recent foundational perspectives on planning and acting (Ghallab *et al.*, 2014) and deliberation functions (Ingrand & Ghallab, 2014). We present a motivating example (§2), provide background (§3), detail our model and two instantiations (§4), provide a proof of minimal agency (§5), introduce goal memory (§6), define the GR problem (§7), and describe its application to three robotics-related tasks (§8). We integrate our discussion of related work throughout the paper, although this does not constitute a thorough survey on goals in the literature on planning, robotics, agents, and actors.

Our projects span GR actors at the coach, team, and single system levels. Like other research on robotics, a cross-cutting concern is eliciting robotic behavior that is consistent, reliable, trustworthy, verifiable, explainable, and predictable. We invite researchers and practitioners to join our ongoing dialog on the topic of goal reasoning.

2. Waypoint Navigation Example

Figure 1 depicts a simple waypoint navigation task where the robot's goal is `at(y)`. This example could apply to controlling an underwater or micro-aerial vehicle in the context of water currents or wind, respectively, as shown as vectors. Dashed curves indicate the bounds of the expected trajectory (i.e., a soft constraint) while the outer box represents a hard constraint that the actor should avoid violating. The actual trajectory of the robot is given by the solid arc that starts at `x` and ends at `y`. The deviating path is due to the difference between the expected flow (dashed vectors) and actual flow (solid vectors).

From the example it is evident that we assume a dynamic environment, exogenous events, and interruptible actions. We also assume that both the environment and the actor's actions have temporal extent.

Below the plot is a representation of the vehicles timeline, which is inspired by the work of Smith et al. (2000). The time window of the plan indicates that the plan should start executing no earlier than the earliest start time (i.e., the leftmost vertical bar) and finish by the latest finish time (the rightmost vertical bar). The large block in the middle indicates the expected transit duration. Inside the timeline

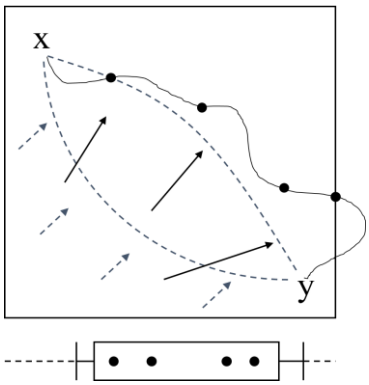


Figure 1: Navigating from `x` to `y`

are dots corresponding to what we call a "notable event," which we define as an event (usually resulting in a state change) that impacts the agent in some way. For this example, the notable events correspond to soft and hard constraint violations; the first two points indicate where the vehicle violates the preferred trajectory while the last two points indicate eminent and actual violation of the hard constraint. These points express possible places and where a decision must be made concerning vehicle behavior.

A GR actor may resolve (i.e., respond to) these events using various strategies (e.g., adjust its expectations, adjust or replace its plan, adjust or change its goal). GR actors differ in the strategies they can apply and how they apply them. In §4, we will use this example to illustrate some of these strategies, after providing some background.

3. Preliminaries

The models and algorithms used for planning in robotics is staggering (e.g., (LaValle, 2006; Ghallab *et al.*, 2014)). We will show how GR can be viewed as a process of refining the constraints on goals. This perspective synthesizes and unifies planning for robotics. First, we define *goals* and review planning as refinement search.

3.1 Goals

Our robotics control applications focus on achieving *goals*, which we define as states an actor desires to achieve (or maintain). To define states, we leverage some elements of the classical planning formalism by Ghallab et al. (2004). Let L be the language for representing world states, $S \subseteq 2^L$ be the set of world states, and $g \subseteq L$ be a goal. Then $S_g = \{s \in S | g \subseteq s\}$ represents the set of states that achieve g. The world is assumed to exist externally to the actor, and a plan is a sequence of actions for transforming an initial state S_0 into S_g. For much of our discussion we focus on a single goal, but the use of a goal set is appropriate for some applications; the model easily extends to goal sets.

We assume that a set of temporal, resource, and ordering constraints apply to goals as well as states and actions, but the exact nature of these constraints is a design decision. Researchers have used a variety of ways to represent such constraints (e.g., as a constraint satisfaction problem (Scala, to appear), in PDDL (Vaquro, to appear), or NDDL (Rajan et al. 2013)). We also assume that completing (or maintaining) goals has intrinsic value for an actor; we return to this assumption in §6 and §7.

For GR to occur the actor must perform actions for transitioning among *both* its external and internal state. To exemplify these, a goal to achieve external state in Figure 1 might be `at(y)`, while a goal to satisfy internal state might be `finished(at(y))`. So we expand and partition the language of the GR actor into $L_{GR} = L_{external} \cup L_{internal}$. We similarly partition the set of goals into $S_g = E_g \cup I_g$. In $L_{external}$ the actor selects actions to achieve E_g. In $L_{internal}$ the actor selects actions to achieve I_g and some internal actions may be conditioned on external goals. *Primitive goals* cannot be decomposed into subgoals.

3.3 Planning as Refinement Search

Our model's theoretical foundation builds from Planning as Refinement Search (Kambhampati, 1994; 1997; Kambhampati *et al.*, 1995), which models planning in a generic way to distinguish planners by their design choices and facilitate their comparison. Refinement planning employs a split and prune model of search, where plans are drawn from a candidate space K. Let a search node N be a constraint set that implicitly represents a candidate set drawn from K. Refinement operators transform a node N_i at layer i into m children $\langle N_{j1}, N_{j2,}, \ldots, N_{jm} \rangle$ at layer $j = i + 1$ by adding constraints which further restrict the candidate sets in the next layer. If the constraints are inconsistent then the candidate set is empty. The authors initially provided two kinds of constraints that can be added by refinement operators: (1) *interval constraints* ensure a variable (i.e., a proposition) maintains a property (i.e., it remains true, false, or unchanged) over an interval; and (2) *truth point constraints* ensure a variable is true at a specific point in time. Kambhampati (1994) conjectured that these constraints could be extended to include behavioral constraints or desires. Kambhampati and Srivastava (1995) extend plan refinement to state-space planning with *contiguity constraints* ensuring, for two actions i and j, no new action can intervene.

Let $N_\emptyset$ represent an initial node whose candidate set equals K and results from only the initial constraint set provided in the problem description (from the perspective of the search process, the refined constraints are empty, thus the subscript $\emptyset$). The REFINEPLAN algorithm begins with $N_\emptyset$ and recursively applies refinements to add constraints until a solution is found. A desirable property of refinements is that each layer of search results in smaller candidates subsets as REFINEPLAN proceeds. Thus the

constraints aid search by identifying inconsistent nodes and providing backtracking points. An optional step is to apply forward consistency checking to further prune candidate nodes, usually at considerable computational cost. Instantiations of REFINEPLAN correspond to different versions of classical planning.

The original model of refinement planning focused on Partial Order Planning, but extensions to the kinds of constraints allowed the refinement framework to incorporate other forms of planning and clarify issues in the Modal Truth Criterion (Kambhampati & Nau, 1994). Unfortunately space limitations prevent a full exposition of these ideas. Briefly, plan refinement allows us to equate different kinds of goal decomposition methods in plan-space and state-space planning. More recent formalisms such as Angelic Hierarchical Plans (Marthi *et al.*, 2008) and Hierarchical Goal Networks (Shivashankar *et al.*, 2013) can also be viewed as leveraging this concept. The focus on constraints in plan refinement also allows a natural extension to the many integrated planning and scheduling systems that use constraints for temporal and resource reasoning.

4. Goal Reasoning as Goal Refinement

To build on plan refinement, we distinguish between a GR process and the actor that is running it because there may be additional processes in the actor such as meta reasoning, learning, etc. We assume a goal is achieved through the execution of some expansion (i.e., plan). A GR process *refines a set of goal nodes G until they can be achieved through execution*. For a goal $g \in G$, a goal node $N^g = \langle C_g, X_g, x, o \rangle$ is a tuple of constraints C_g, possible expansions that could achieve the goal X_g, the currently selected expansion $x \in X_g$, and goal lifecycle mode o. A GR process begins with $N^g_\emptyset$, which consists of the candidate space of all possible executions achieving g, and makes decisions that refine its goals S_g or I_g through a series of refinements R on goal nodes until it selects one expansion $x \in X$ for execution. Similar to plan refinement, goal refinement takes a Least Commitment (Weld, 1994) approach. Further, planning and learning (Veloso *et al.*, 1995) could be incorporated in certain parts of the system, but learning is not a requirement for goal refinement. Defining GR as goal refinement allows us model GR in a generic way to distinguish planners by their design choices and facilitate their comparison

4.1 Constraints (C_g)

We partition the constraints $C_g = C_g^{given} \cup C_g^{added}$ into (1) those given to the GR process from the process that invoked it (e.g., human operator, meta-reasoning process, coach) and (2) those that it adds during refinement.

Constraints can be temporal (finish by a certain time), ordering (do `x` before `y`), maintenance (remain at a certain depth), resource (consider only one goal at a time), or computational (only use so much CPU or memory). Top-level constraints can be pre-encoded or based on drives (e.g., (Coddington *et al.*, 2005; Young & Hawes, 2012)). Hard constraints in C_g must be satisfied at all times (stay within the box in the waypoint example), while soft constraints should be satisfied if possible (follow a preferred trajectory).

4.2 Expansions (X_g)

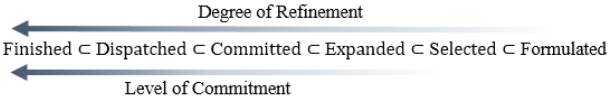
To distinguish goal refinement from planning, we define the set of possible *expansions* X_g as the means of achieving g. A goal S_g or I_g can be achieved by executing an expansion. We call $x \in X_g$ a *selected expansion* that is currently slated for execution.

The term "expansion" highlights GR as any process that performs goal refinement. Planning (in the classical sense) is one kind of expansion, but not all possible expansions of S_g are plans. Robotics systems are often integrated in layers that operate on distinct granular models of the world. So expansions can include, but are not limited to: a simple rule, a richly detailed goal or task network, a (state-based) plan, the switching of behaviors, a change in parameters for an adjoining deliberation layer, an algorithm for learning new or revising existing knowledge, a recipe for unlearning, or, for a team of robots, a specially crafted algorithm.

Expansions can be provided at different times to an actor: *designed* expansions are declared as part of the actor's specification (i.e., as part of L_{GR}), while *learned* expansions are new ways of problem solving derived from the actor's experience and investment in capturing or revising new knowledge (i.e., L_{GR}, which may grow or change to incorporate this new knowledge). This knowledge can be captured online during execution, during the actor's deliberation, or offline prior to execution.

4.3 Modes (o)

The modes of a goal indicate successively smaller candidate sets towards eventual execution. We label each set with a mode from {Formulated, Selected, Expanded, Committed, Dispatched, Finished}. Transitions between these modes lead toward eventual execution (Clement *et al.*, 2007). We present two views of these modes. In the goal refinement view, each mode can be a strict subset of the next in the candidate space of goals:



Reading this from right to left (due to the subset relation): After an actor formulates and selects a goal, planning involves searching through candidate expansions and

selecting one. Finally, the dispatch step dispatches a plan for execution, monitors its trajectory (making corrections as needed), and marks the goal as achieved if execution went well. Each transition to a new mode reduces the candidate set, increases the level of commitment the actor has made to a goal, and increases its degree of refinement. Refinements follow naturally from the view of actors that are performing deliberation (Ingrand & Ghallab, 2014). We invest the next section discussing a left-to-right view of the modes.

4.4 The Goal Lifecycle

The lifecycle for a goal (Figure 2) captures the possible decision points of a GR actor and complements a plan's lifecycle (e.g., (Pollack & Horty, 1999; Myers, 1999). Decisions consist of applying a *strategy* (denoted using an arc in Figure 2; **boldfaced** in this section) that transitions a goal among *modes* (denoted using large or small rounded boxes) in the lifecycle. The *g*'s in the goal lifecycle correspond to goals and *x*'s correspond to expansions. Transitions are verbs and modes qualify a goal's mode (e.g., *select*(S_g) transitions S_g from *formulated* to *selected* mode).

Goals in an *active* mode are those that have been formulated but not yet dropped. The **formulate** strategy is the first decision point. In many actors, this decision is carefully (implicitly, statically) encoded (e.g., an observation may fire a trigger to achieve some goal). Vattam *et al*., (2013) describe goal formulation strategies. The **drop** strategy causes a goal to be "forgotten" and can occur from any active mode. This decision can be sophisticated and involve learning from the execution or attempted expansion of a selected goal. To **select** a goal indicates intent. A selection strategy depends on which goals were formulated and effects the selection of a goal for further refinement. The **expand** strategy decomposes a goal into subgoal(s) or a primitive goal. The **commit**($\boldsymbol{x}$) strategy chooses the best expansion for execution; we assume domain-specific quality metrics can assess expansion quality. The **dispatch**($\boldsymbol{x}$) strategy slates the best expansion for execution and places the goal in an *executing* mode. In both single- and multi-agent systems, a plan may undergo further refinement (i.e., scheduling) prior to execution. Prior refinements could favor a least commitment approach on temporal/resource constraints to allow for flexible dispatch (Conrad *et al.*, 2009).

Because we assume an online dynamic world, goals in an executing mode are subject to transitions that result from expected or unexpected external state changes during execution. The **monitor** strategy can be passive (i.e., do nothing) or proactive (i.e., monitor periodically) while a goal is dispatched. As long as its plan's execution does not encounter any notable events, goal execution follows a normal path toward achievement. If no notable events occur and the dispatched expansion completes, then the **finish** strategy marks the goal as *finished*, which may store the goal to aid future deliberation. Not all goals may reach this mode because goals can be dropped.

When notable events occur, the **evaluate** strategy determines how they impact goal execution (positively or negatively). An *evaluated* mode does not imply that execution of the current expansion is stopped. If the evaluation does not impact the goal, it can **continue** with the execution. However, if the event impacts the current goal, the set of **resolve** strategies define a suite of paths toward goal achievement. An obvious choice is to change the world model using **adjust**($\boldsymbol{L}$), but adjusting its model does not resolve the current goal and further refinements are required. In the waypoint example, we can imagine that the GR actor could decide to adjust the bounds on the preferred trajectory using the variance of the actual path. However, this is only an "internal" adjustment, and it does not subsequently communicate the adjusted expectations to its execution system. The GR would simply ignore any future bounds violations that fall within the new expectations. Rather than adjust its expectations, the GR might apply a **repair**($\boldsymbol{x}$) strategy by repairing $\boldsymbol{x}$ so that it meets the new context; this is the so-called replanning approach. In the waypoint example, this might involve communicating new bounds, allowing more time, or selecting behaviors more appropriate to the current conditions. This resolution repairs but does fundamentally alter the current plan. If no repair is possible (or desired) then the GR can apply the **re-expand**($\boldsymbol{g}$) strategy to reconsider the original plan from scratch. In the

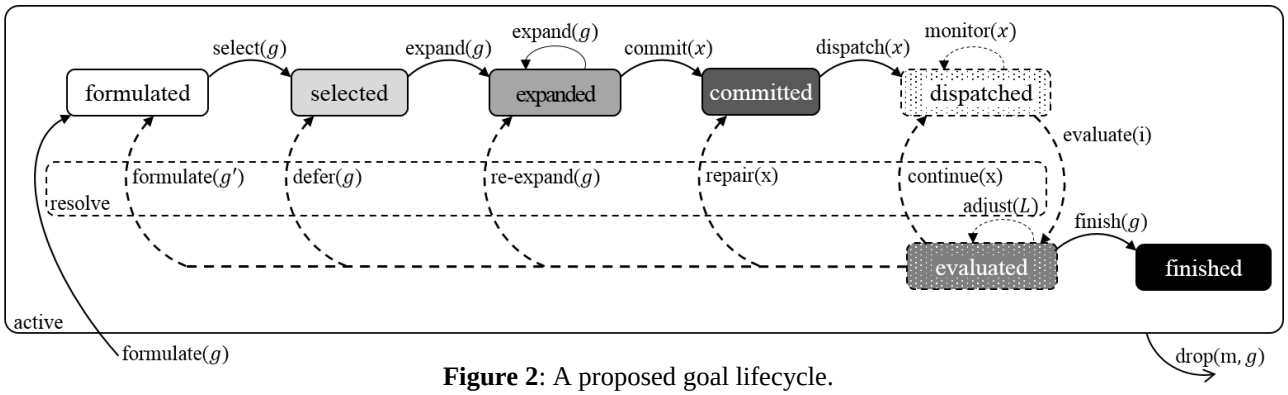


Figure 2: A proposed goal lifecycle.

waypoint example, this might occur if an obstacle is found to exist en route. The **defer**(g) strategy instead postpones the goal for further processing. In the example, this may happen if current conditions are deemed as unfavorable for achieving g, but the GR decides to retain g as worth pursuing in the future. A final option occurs in **formulate**(g'), which abandons g in favor of a newly formed goal g'.

We very recently discovered the work on goal lifecycles in the Autonomous Agents literature that is closely related to the goal lifecycle proposed in this paper. Harland et al. (2014) extend their earlier work (Thangarajah et al., 2010) to present a goal lifecycle for BDI agents, provide operational semantics for their lifecycle, and demonstrate the lifecycle on a Mars rover scenario. Work by Winicoff et al. (2010) has also linked Linear Temporal Logic to the expression of goals that the project we discuss in §8.3 may be able to leverage. We plan to fully explore these approaches in future work.

4.5 Strategies and Strategy Composition

Strategies can be simple or complex. For example, *select*(g) could be implemented as a simple rule (e.g., automatically select any goal that is formulated) or it could be implemented as a learned policy that considers knowledge about the environment, which goals are currently executing, and their priority. Similarly, the *drop* strategy could be very simple (e.g., in Figure 1, drop any goal when hard constraints are violated), or the *drop* strategy could attempt to learn long-term knowledge from the information gathered in the goal node as it transitioned through the life cycle (e.g., in Figure 1, adjust future expectations for this region to account for greater flow).

Strategies can be composed, of which the *resolve* strategy in Figure 2 is one example. A composition representing classical planning demonstrates goals that are formulated by an external process to the actor. Let g_2 be a goal and FINDPLAN be a planning algorithm producing one expansion x that equates to a plan for achieving g_2. Then a classical planning strategy is composed:

$$\textit{plan}(g_2) \Rightarrow \text{formulate+select}(g_2),\ x = \text{FINDPLAN}(g_2).$$

Another strategy composition can relate the goal lifecycle to partial satisfaction planning (PSP) and soft goals for planning (Benton et al., 2010). PSP is closely aligned with GR for the project discussed in §8.2. In that framework planning proceeds on an *oversubscribed set* of goals where a penalty is assessed for not meeting goals; the planner constructs a plan that maximizes the utility (i.e., the payoff) while minimizing the penalty. Let G_3 be an oversubscribed goal set and PSPPLAN be a planning algorithm that selects the subset of goals and a plan x to achieve them given the

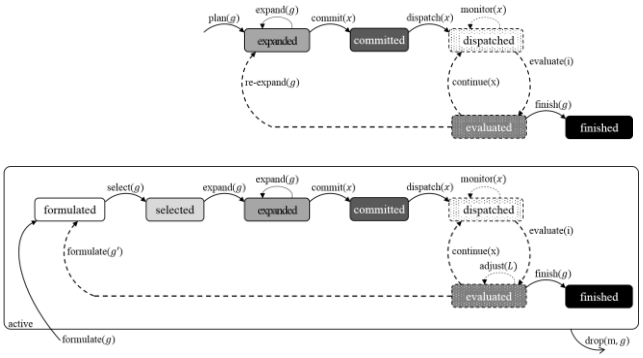


Figure 3: Contrasting replanning (top) with GDA (bottom) instantiations of the GR model.

objective criteria. Then a PSP planning strategy is composed:

$$\textit{psp-plan}(G_3) \Rightarrow \text{formulate}(G_3),\ x = \text{PSPPLAN}(G_3)$$

It may seem *psp-plan*(G_3) is identical to *plan*(g_2), however PSPPLAN(G_3) performs goal selection, expansion and plan selection, while FINDPLAN (g_2) only performs expansion and plan selection.

4.6 Instantiations of Goal Reasoning

We next demonstrate how the full GR model can instantiate two existing GR systems. The first is a replanning system, which is a common approach to solving online dynamic planning (e.g., (Yoon *et al.*, 2007)). Figure 3 (top) shows how the GR model can instantiate such replanning systems. As shown, formulation is presumed (designed) in this model and we have used the "plan(g)" composition of §4.5.

Figure 3 (bottom) shows one instantiation of the GR model for Goal-Driven Autonomy (GDA); GDA agents perform online planning and execution (Klenk *et al.*, 2013). A GDA agent consists of an intelligent controller that not only interacts with a planner and the execution environment, but also includes components for GR, separating the planning process from those for goal formulation and management. The controller takes as input an initial planning problem and sends it to the planner. The planner returns (1) a plan, which is a sequence of actions, and (2) a corresponding sequence of expectations consisting of the states expected to result after executing each action in the plan. The controller *dispatches* the plan's actions to the execution environment and then runs a sophisticated *evaluate* strategy. While a plan is *dispatched*, the controller *evaluates* new information by (1) comparing observations with expectations to discover discrepancies. For a discrepancy, it (2) may generate an explanation, which takes the discrepancy plus a history of past actions and observations to propose one or more possible causal

explanations. Depending on the explanation, the controller (3) may *adjust(L)* to correct its model or expectations of the world, *formulate* a new goal in response to the discrepancy, and then *drop* the existing goal. Finally, (4) this new goal moves through the goal lifecycle (*selected*, *expanded*, etc.) and is weighed against other goals for execution.

4.7 Instantiations of Strategies

We now detail two possible instantiations of the strategies for the GDA architecture from Figure 3 (bottom). The M-ARTUE system (Wilson et al, 2013) takes a direct approach to the goal formulation step, in which all possible goals are considered by the agent according to a set of domain-independent heuristics that assess a goal's fitness in three different dimensions: social, exploration and opportunity. The fitness of each goal in these three dimensions is weighted by the urgency of each of the respective needs to come up with a single score by which all goals are compared. A preliminary study of M-ARTUE showed that it reached performance comparable to the use of domain-specific knowledge for guiding goal selection in a test domain.

The FoolMeTwice agent (Molineaux & Aha, to appear) employs a monitor and integrate strategies that incorporate the environment to effect the agent's knowledge of what events are possible. The FoolMeTwice agent is "surprised" when it cannot explain (i.e., find a history of events consistent with) its observations of the environment. When surprised, FoolMeTwice hypothesizes a new model of a previously unknown event that caused its surprise. These models are thereafter use both monitor and integrate strategies to improve the agent's performance at these tasks.

5. A Proof of Minimal Agency

Decision is a key element of agency. The degree to which the actor's decisions depend on dynamic deliberation rather than pre-encoded knowledge determines its *agency*. Clearly, if a single rule (i.e., a design) covers all contingencies for any strategy that arises during execution, then the actor need not make a decision. Decision points are noteworthy because – assuming the agent acts on a single external goal E_g – the actor's goal switches from achieving E_g to achieving some internal decision goal $I_d \in I_g$. Agency is an increasing function of the number and kind of decision goals. Further, we can prove:

Proposition 1: *The number of primitive active goals for a deliberative agent must be at least two, one of which must be a decision goal, I_d.*

Proof sketch: We can reason about any transition in the lifecycle without loss of generality. Suppose the actor must decide on one of two possible paths for $select(S_g)$ for a primitive goal S_g. There are two cases to consider regarding whether the agent deliberates on this decision. Let a *rule* be a provided (or learned) sequence of steps to achieve some goal. (1) Suppose the actor applies a rule to decide (even non-deterministically), then the agent did not deliberate, which contradicts our assumption of a deliberative actor. (2) If there is no such rule, then the agent must formulate an internal decision goal $I_d =$`is-selected-`S_g (i.e., the mode of S_g is *selected*) and switch to the goal `is-achieved-`I_d. We can imagine many different ways the actor might achieve this new goal (e.g., it can explore to generate knowledge or exploit its existing knowledge). At this point, we have already shown that at least two primitive goals were required. *Q.E.D.*

Corollary 1: *An actor will be unable to resolve a decision goal if it tries to deliberate without enough designed (or learned) knowledge.*

Corollary 2: *The number, kind, and frequency of primitive decision goals defines a spectrum of deliberation and agency for actors.*

6. Goal Memory

In addition to moving goals through a lifecycle toward achievement, a decision-making actor assesses tradeoffs between various criteria (e.g., priority, domain-specific quality functions, global utility, long term payoff). We use the goal lifecycle in conjunction with a *goal memory* to characterize a goal management process that simultaneously addresses both.

Our use of the term *goal memory* here is distinct from its typical use in cognitive goal memory. In cognitive science, goal memory is typically discussed as a mental construct with representations and processes that are used to store and manage goal-related requirements of the task that a cognitive agent happened to be engaged in. While issues such as interference level, strengthening and priming constraints are key requirements to mimic human memory (Altmann & Trafton 2002), we ignore any such considerations because we are not concerned with the cognitive plausibility of our model of the goal memory.

Figure 4 shows the $m \times n$ goal memory in a table M, where a cell M_{ij} represents the i^{th} goal g_i $(1 \leq i \leq m)$, its j^{th} quality criteria $q_i (1 \leq j \leq n)$, and mode. We describe these criteria in the following paragraphs.

The *priority* criterion of a goal determines its importance relative to other goals. Let $f_{priority}(g_i)$: $g_i \rightarrow \mathbb{I}$, then $M_{i1} = f_{priority}(g_i)$. Priority depends on the agent's current state, and so may change dynamically.

	Priority	Inertia	Mode	Quality Metrics		
	q_1	q_2	q_3	q_4	…	q_n
g_1						
g_2						
…						
g_m						

Figure 4*:* A Goal memory of m goals and n quality metrics.

The *inertia* criterion of a goal g_i characterizes the strength of bias against changing its current mode because of prior commitments. Let $f_{intertia}(g_i) : g_i \rightarrow \mathbb{I}$, and let $M_{i2} = f_{inertia}(g_i)$. Inertia is defined as a function of g_i's mode, its number of expansions $|X_g|$, its *staleness* (i.e., the number of time steps since it last transitioned), the number of transitions that have been applied to g_i, and the resources committed to g_i.

The *mode* criterion of a goal g_i determines its relative importance based on how far along it is in the goal lifecycle. For instance, if a goal is closer to execution, it has a higher value because we want to move goals toward finished. Let $f_{mode}(g_i)$: $g_i \rightarrow \mathbb{I}$, then $M_{i3} = f_{mode}(g_i)$.

The remaining criteria express the quality (e.g., cost, value, risk, reward) of achieving g_i with the currently selected expansion $x \in X_g$. These are domain-specific quality metrics, and we provide some examples when discussing applications in §8. These metrics may also include domain-independent quality metrics such as minimizing makespan (i.e., parallel execution time) or minimizing the plan length (i.e., number of plan steps).

In addition to the aforementioned criteria, we conjecture that additional book-keeping columns may be necessary. These include but are not limited to constraints, alternative expansions, parent of goal, type of goal, etc.

7. The Goal Reasoning Problem

A GR agent examines its goal memory state M_t at time t and chooses a strategy that maximizes its long-term rewards using $\sum_t^\infty \gamma^t rew_t$, where γ^t is a discount factor and rew_t is the agent's reward at t, which we model as $rew: M \times R \rightarrow \mathbb{R}$.

For our formal model, we make some simplifying assumptions that are too limiting for integrated robotics but aid in explaining the model. In addition to the dynamic environment and interruptible actions already assumed, we assume for the exposition of the model:

- **Markovian dynamics:** The current choice for an actor is based only on its last (known) state.
- **Infinite horizon with discount:** The actor is myopic; it considers distant future states to be less important than eminent future states, and may favor locally optimal solutions that are globally suboptimal.
- **Non-deterministic actions**: An actor's action may have multiple possible outcomes.

Under these assumptions, we can model GR as a *Markov Decision Process* (MDP), given that the transition function and the reward function are known. Given a current goal memory state M, a GR strategy $r \in R$ and a next goal memory state M', the mode transition function $T(M, r, M')$ is the probability of transitioning from M to M' using strategy r. For MDPs, there exists an optimal deterministic stationary policy (Kaelbling *et al.*, 1996), implying the existence of an optimal value function for a current goal memory state M:

$$V^*(M) = max_{policy}\left(E\left(\sum_{t=0}^{\infty} y^t\, rew_t\right)\right),$$

where $(0 \leq \gamma < 1)$ is the discount factor. This optimal value function is unique and reduces to $\forall M' \in r(M)$:

$$V^*(M) = max_r(rew(M,r) + \gamma \sum_{M'} T(M,r,M')V^*(M')).$$

Given this, we can specify the optimal policy as:

$$policy^*(M) = \arg max_r\,(rew(M,r) + \gamma \sum_{M'} T(M,r,M')V^*(M')).$$

If T or rew are unknown, then GR can be modeled as a reinforcement learning (Sutton & Barto, 1998) problem, where deliberation results in a learned policy. Reinforcement learning is a rich area of research that is out of scope for this paper.

8. Applications of Goal Reasoning

Our group is working on two robotics and one simulated robotics projects involving GR. We review these with a focus on the expected value added by using GR, our technical approach, and the GR research questions we are addressing.

8.1 Unmanned Underwater Vehicle (UUV) Control

UUVs have been used for tasks such as inspection of underwater structures (Antonelli et al., 2001), mine countermeasures (LePage & Schmidt, 2002), and scientific observation (Binney et al., 2010). These have engendered work on motion planning (e.g., Tan et al., 2004), which can guide vehicles to desired locations but cannot select goals. These missions have short duration (at most eight to sixteen hours) and operate over a small region.

Long-duration missions, potentially lasting weeks or months over much larger regions, present new challenges for guidance systems, as the ocean environment is unpredictable and partially observable. A UUV on a long-duration mission must react competently to notable objects and events. It may need to change its objectives or even abort its mission due to unforeseen environmental hazards, underwater barriers, encounters with other vehicles, or

failures of onboard systems. A common approach in the face of a dynamic environment would be replanning. Cashmore et al (2013) confront the need for long-duration autonomy in UUVs and examine the problem of modeling motion for task-level mission planning. Their architecture reacts to notable events (observations of the environment that differ from assumptions) by remodeling the environment and replanning for a fixed set of goals. In the language of our GR model, their approach applies the *adjust(L)* strategy followed by the *re-expand(x)* strategy. This is a case where *re-expand(x)* equates to replanning.

An alternative approach could allow the UUV to *regoal*. Consider a UUV taking oceanographic measurements (e.g., water salinity) when it detects a nearby surface vessel. While motion planning systems will likely continue the measurement task, minimizing risk of collision while maximizing data quality, they cannot consider the broader implications of the vessel's arrival and how best to respond. Depending on the location, nature of the mission, and the identity of the approaching vessel, the UUV may need to communicate with it, attempt to avoid detection, or abort the data-collection mission and return to notify its operator of the surface vessel's approach. An at-sea UUV has limited communication with human operators, and must make such goal decisions autonomously.

To provide a UUV with the ability to reason about and dynamically select goals while pursuing long-term missions, we are applying GDA to guide a Bluefin underwater vehicle, initially in simulation but with planned execution on a real vehicle. GDA can generate appropriate goals in response to unplanned situations and is therefore well-suited to the control of unmanned vehicles at sea.

We use MOOS-IvP (Benjamin et al., 2010) to provide reactive navigation guidance. MOOS is a message-passing system with a centralized publish-subscribe model. IvP Helm is a behavior-based MOOS application that chooses desired heading, speed, and depth for the vehicle in a reactive manner to generate collision-free trajectories. IvP Helm uses an interval programming technique that optimizes over an arbitrary number of objective functions to generate desired navigation values. The GDA agent complements the IvP Helm's reactive behaviors by enabling the capacity for deliberative reasoning for longer missions. Thus, we use the GDA agent to perform GR, IvP Helm to provide navigation guidance, and Bluefin's Huxley control architecture for low-level control.

8.2 Unmanned Air Vehicle (UAV) Control

UAVs have been used frequently in military operations, controlled via teleoperation in surveillance and targeting missions, for example. As they become more autonomous they will also be deployed in air combat operations in areas that are highly dynamic, uncertain, and adversarial. In such environments, UAVs will have to coordinate with manned aircraft, which the USA military highlights as a critical technical challenge (DoD, 2013). Our project's objective is to develop and demonstrate the utility of a GR agent for controlling simulated UAVs in manned-unmanned air combat teams, where the teamed pilots will manage the UAVs' activities.

The air combat environment is highly complex with stochastic, dynamic, adversarial, and partially observable elements. Highly autonomous decision making in such an environment requires agents to respond to situations for which they lack pre-programmed responses. The UAVs cannot rely solely on the pilot for constant oversight in these situations because they must pilot their own vehicle.

For this task, we are integrating a novel GR agent in a decision-making system called the TBM (Tactical Battle Manager), which should advance the state-of-the-art in several respects. This high-tempo environment requires decisions to be made within seconds as indecision could lead to loss of human life or destruction of expensive assets. The human pilot must specify goals and preferences. Finally, scenarios will consist of multiple vehicles, and actions by the actor effect the pilots and other UAVs' agents.

Our GR model is inspired by Young and Hawes's (2012) model, in which *desires* are satisfied via a system of *drives*. At any given time the desire monitor and state monitor inspect the world state. If an event agitates a desire, then the GR may formulate a new goal. For example, suppose a manned vehicle in a manned-unmanned team of vehicles was just shot down. This event would agitate an ENSURE HUMAN SAFETY desire in the actor controlling the UAV, and a drive would then formulate a DEFEND CRASH SITE goal.

To evaluate GR we will use two modern air combat simulations, namely the Next Generation Threat System (NGTS) (2013) and the Analytic Network for Network-Enabled Systems (AFNES). We have integrated a simple GR agent with NGTS; it replaces plans to control air vehicles when notable events occur. We will apply GR to a set of simulated scenarios (with random variations) and measure mission success with and without goal reasoning.

We hypothesize that the integration of GR in TBM will increase the performance of mixed teams in air combat missions, and that GR will reduce the amount of oversight that pilots must provide to their UAV teammates, because they will be able to reason and respond to unexpected situations as they occur.

8.3 Control for Collaborative Sensing

Between the time of a tragic disaster (e.g., the Philippines Typhoon) and the arrival of support operations, emergency response personnel need information concerning the whereabouts of survivors, the condition of infrastructure, a suggested ingress and evacuation routes. Current practice

for gathering this information relies on drone operators and human pilots of helicopters. We believe a hetealrogeneous team of autonomous vehicles with sensor platforms can automate many parts of the information gathering, thus freeing humans to perform more critical tasks and improving the response time for Humanitarian Assistance/Disaster Relief operations.

Planning trajectories for teams *a priori* to achieve a single objective requires solving a high dimensional optimization problem (Yilmaz *et al*., 2008) to compute optimal trajectories that are tightly coupled to the initial assumptions/goal. Bio-inspired and other reactive guidance strategies simplify this problem by using more goal-directed behaviors for area coverage (Liu & Hedrick, 2011) and discrete target tracking (Haque *et al.*, 2008; Kruecher *et al.*, 2007). These behaviors rely on local measurements and instantaneous gradients to guide robots. Still, no behavior or trajectory can handle all contingencies.

A promising approach, inspired by animal behavior, uses finite state automata (FSA) for mobile robot guidance (Balch *et al.*, 2006). Hand-coding an FSA for each execution of a robot is tedious and error prone. Kress-Gazit et al. (2009) instead synthesize an FSA using a Linear Temporal Logic specification (LTL-spec) that views synthesis as searching for a game-theoretic table in which the robot takes actions to achieve its goals against actions taken by the environment (i.e., the adversary). This strategy guarantees correct behavior if the LTL-spec is never violated, but synthesis is exponential in the number of (environmental and sensing) goals. This is clearly intractable for large teams of robots, and we use GR as a "coach" to select goals to maintain tractable synthesis for individuals within the team.

Our technical approach draws on the goal refinement process of decomposing a high-level goal into predicates that then guide LTL-to-FSA synthesis. As the potential number of predicates is large for a multi-vehicle, multi-goal mission, we developed a hierarchical approach that separates GR, planning/scheduling, and vehicle guidance tasks into discrete processes. Our approach converts active goals into an LTL specification for the mobile agents. If a specification is unsatisfiable, an error report is returned to the GR actor.

At runtime, we monitor the agents' progress through their FSAs, and constantly update the GR actor's model of the environment. Notable events occur when the actors: achieve a substantial sub-goal; determine they cannot achieve their current sub-goal; or their FSA does not specify how to respond to an unexpected change in a vehicle's state. Candidate goals are evaluated using an approximate environment and team model based on the MASON (Luke *et al.*, 2005) multi-agent simulator, which can perform some FSA synthesis and apply the optimal or reactive guidance algorithms. These produce the quality metrics the GR agent uses to select which goals to activate.

9. Summary and Future Work

We observed that goal reasoning (GR) occurs when an actor observes notable events and that it falls along a spectrum of design to deliberation. The extent to which an actor takes initiative to deliberate over its goals provides a measure of autonomy. We presented the goal lifecycle to demonstrate how modes act as constraints in the management of goals and discussed how this lifecycle instantiates GR models: for replanning and Goal-Driven Autonomy (GDA). We formalized the GR problem by introducing a goal memory and GR operators, casting the problem of GR in terms of choosing a composition of GR operators to maximize an actor's future rewards. Although our model is general enough to be agnostic about which approach is used for this purpose, we then related the GR problem to a Markov Decision Process (if states and the value function are known) and Reinforcement Learning (if states or the value function are unknown). Finally, we discussed three ongoing robotics-related projects in which we are using a model of GR for decision making and control.

There are many benefits to our proposed GR model:

a. It provides a *common language* for discussing deliberation in actors, and is rich enough to frame the conversation among researchers who study robotics, planning, or scheduling.
b. It is *instantiable*; it covers existing subclasses and can grow to future knowledge/systems.
c. Strategies make the model *composable* and able to incorporate the variety of design decisions of an actor. Strategies can be empty (i.e., no-op), static/dynamic policies, hand-coded (or learned) rules or cases, or domain-specific algorithms.
d. From a software design perspective, the model allows for *rapid prototyping* of systems. A team can begin with handcoded/no-op strategies to determine a platform's viability, which provides a baseline for assessing autonomy. This low-bar approach also aids in focusing knowledge modeling on only the parts of the system where decisions will be made, and thus helps with knowledge engineering for robotics.
e. This model *spans layers of deliberation* at the individual, team, and coach levels.

We will soon extend the formal model of GR, instantiate it in the projects described in §8, and analyze its advantages and limitations by evaluating those actors' performances. We are especially interested in linking the GR lifecycle to recent models of replanning (Talamadupala et al., 2013) and continual planning (Scala, to appear).

We described goal reasoning in terms of a lifecycle that refines an actor's goals and expansions, and summarized its application to robotics-related tasks. Our research is in its early stages, and we invite feedback on this model.

Acknowledgements

The authors for this project were funded by OSD. We also thank the anonymous reviewers whose comments helped improve the paper.

References

Altmann, E. M., & Trafton, J. G. (2002). Memory for goals: An activation-based model. *Cognitive Science*, 26, 39-83.

Antonelli, G., Chiaverini, S., Finotello, R., & Schiavon, R. (2001). Real-time path planning and obstacle avoidance for RAIS: Aan autonomous underwater vehicle. *IEEE Journal of Oceanic Engineering*, *26*(2), 216-227.

Balch, T., Dellaert, F., Feldman, A., Guillory, A., Isbell, C.L., Khan, Z., Pratt, S.C., Stein, A.N., & Wilde, H. (2006). How multirobot systems research will accelerate our understanding of social animal behavior. *Proceedings of the IEEE*, *94*(7), 1445-1463.

Benton, J., Do, M., & Kambhampati, S. (2009). Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence*, *173*(5-6), 562–592.

Benjamin, M., Schmidt, H., Newman, P., & Leonard, J. (2010). Nested autonomy for unmanned marine vehicles with MOOS-IvP. *Journal of Field Robotics*, *27*(6), 834-875.

Binney, J., Krause, A., & Sukhatme, G.S. (2010). Informative path planning for an autonomous underwater vehicle. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, (pp. 4791-4796). Anchorage, AK: IEEE Press.

Chien S., Knight R., Stechert A., Sherwood R., and Rabideau, G. (2000) Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. *Proceedings of the Conference on Automated Planning and Scheduling* (pp. 300-307). Menlo Park, CA: AAAI Press.

Cashmore, M., Fox, M., Larkworthy, T., Long, D., and Magazzeni, D. (2013). Planning Inspection Tasks for AUVs. In *Proceedings of MTS/IEEE OCEANS 2013*. San Diego, CA: IEEE Press.

Clement, B.J., Durfee, E.H., & Barrett, A.C. (2007). Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, *28*, 453–515.

Coddington, A.M., Fox, M., Gough, J., Long., D., & Serina, I. (2005). MADbot: A motivated and goal directed robot. *Proceedings of the Twentieth National Conference on Artificial Intelligence* (pp. 1680-1681). Pittsburgh, PA: AAAI Press.

Conrad, P., Shah, J. & Williams, B. (2009) Flexible execution of plans with choice. *Proceedings of the Conference on Automated Planning and Scheduling* (pp. 74-81). Menlo Park, CA: AAAI Press.

DoD (2013). Unmanned systems integration roadmap: FY2013-2038 (Reference Number 14-S-0553). Department of Defense, Washington, DC.

Ghallab, M., Nau, D.S., & Traverso, P. (2004). *Automated planning: Theory and practice*. San Mateo, CA: Morgan Kaufmann.

Ghallab, M., Nau, D., & Traverso, P. (2014). The actor's view of automated planning and acting: A position paper. *Artificial Intelligence*, *208*, 1–17.

Harland, J., Morley, D., Thangarajah, J., & Yorke-Smith, N. (2014). An operational semantics for the goal life-cycle in BDI agents. *Autonomous Agents and Multi-Agent Systems*, *28*(4), 682–719.

Haque, M., Rahmani, A, & Egerstedt, M. (2010). Geometric foraging strategies in multi-agent systems based on biological models. In *Proceedings of the 49th IEEE Conference on Decision and Control*. Atlanta, GA: IEEE Press

Ingrand, F., & Ghallab, M. (2014). Robotics and artificial intelligence: A perspective on deliberation functions. *AI Communications*, *27*(1), 63-80.

Kaelbling, L.P., Littman, M.L., & Moore, A.P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237-285.

Kambhampati, S. (1994). Design tradeoffs in partial order (plan space) planning. *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (pp. 67-97). Chicago, IL: AAAI Press.

Kambhampati, S. (1997). Refinement Planning as a unifying framework for plan synthesis. *AI Magazine*, *18*(2), 67-97.

Kambhampati, S., Knoblock, C.A., & Yang, Q. (1995). Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, *76*, 168-238.

Kambhampati, S. & Nau, D. (1994). On the nature of modal truth criteria in planning. *Proceedings of the 12th National Conference on Artificial Intelligence* (pp. 67-97). Seattle, WA: AAAI Press.

Kambhampati, S., & Srivastava, B. (1995). Universal classical planner: An algorithm for unifying state space and plan space planning. *New Directions in AI Planning* (pp. 261-271). IOS Press.

Klenk, M., Molineaux, M., & Aha, D.W. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, *29*(2), 187-206.

Kress-Gazit, H., Fainekos, G.E., & Pappas, G.J. (2009). Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, *25*(6), 1370-1831.

Kreucher, C.M., Hero, A.O., Kastella, K.D., & Morelande, M.R. 2007. An Information-Based Approach to Sensor Management in Large Dynamic Networks. *Proceedings of the IEEE* 95(5), 978-999

LaValle, S., M. (2006). *Planning Algorithms*, Cambridge University Press.

LePage, K.D., & Schmidt, H. (2002). Bistatic synthetic aperture imaging of proud and buried targets from an AUV. *Journal of Ocean Engineering*, *27*(3), 471-483.

Liu, S-Y., & Hedrick, J.K. (2011). The application of domain of danger in autonomous agent team and its effect on exploration efficiency. In *Proceedings of the IEEE American Control Conference.* San Francisco, CA: IEEE Press.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, *81.7*(2005), 517-527.

Marthi, B, Russell, S., & Wolfe, J. (2008). Angelic hierarchical planning: Optimal and online algorithms. *Proceedings of the International Conference on Automated Planning and Scheduling* (pp. 222-231). Menlo Park, CA: AAAI Press.

Molineaux, M., and Aha, D.W. (to appear). Learning Unknown Event Models. In *Proceedings of the Twenty-Eighth AAAI Conference*. Quebec City, Quebec, Canada.

Myers, K.L. (1999). CPEF: A continuous planning and execution framework. *AI Magazine*, *20*(4), 63-69.

NGTS (2013). Next Generation Threat System. [www.navair.navy.mil/nawctsd/Programs/Files/NGTS-2013.pdf]

Pollack, M.E., & Horty, J. (1999). There's more to life than making plans: Plan management in dynamic, multiagent environments. *AI Magazine, 20*, 71-83.

Rajan, K., Py, F., & Barreiro, J. (2013). Towards deliberative control in marine robotics. In *Marine Robot Autonomy* (pp. 91–175). Springer.

Scala, E. (to appear). Continual planning via reconfiguration and goal revision. In *Working notes of the ICAPS Workshop on Planning and Robotics*.

Shivashankar, V., Alford, R., Kuter, U., & Nau, D. (2013). The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. *Proceedings of the 23rd International Joint Conference on Artificial Intelligence* (pp. 2380-2386). Beijing, China: AAAI Press.

Smith, D., Frank, J., & Jonsson, A. (2000). Bridging the gap between planning and scheduling. *Knowledge Engineering Review, 15*, 61-94.

Sutton, R.S., & Barto, A.G. (1998). *Introduction to reinforcement learning*. Cambridge, MA: MIT Press.

Talamadupula, K., Smith, D. E., Cushing, W., & Kambhampati, S. (2013). A Theory of Intra-Agent Replanning. *Working notes of the ICAPS Workshop on Distributed Multiagent Planning*.

Tan, C.S., Sutton, R., & Chudley, J. (2004). An incremental stochastic motion planning technique for autonomous underwater vehicles. In *Proceedings of IFAC Control Applications in Marine Systems Conference* (pp. 483-488). Ancona, Italy: Elsevier.

Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2011). Operational behaviour for executing, suspending, and aborting goals in BDI agent systems. In *Declarative Agent Languages and Technologies VIII* (pp. 1–21). Toronto, Canada: Springer.

Tan, C.S., Sutton, R., & Chudley, J. (2004). An incremental stochastic motion planning technique for autonomous underwater vehicles. In *Proceedings of IFAC Control Applications in Marine Systems Conference* (pp. 483-488). Ancona, Italy: Elsevier.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D. W. (2013, December). Breadth of Approaches to Goal Reasoning: A Research Survey. In *Goal Reasoning: Papers from the ACS Workshop* (p. 111).

Vaquero, T., Nejat, G., & Beck, J.C. (to appear). Planning and scheduling single and multi-person activities in retirement home settings for a group of robots. In *Working notes of the ICAPS Workshop on Planning and Robotics*.

Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, *7*(1), 81-120.

Weld, D.S. (1994). An introduction to least commitment planning. *AI Magazine, 15*, 27-61.

Wilson, M., Molineaux, M., & Aha, D.W. (2013). Domain-Independent Heuristics for Goal Formulation. In *Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference*. St. Pete Beach, Florida.

Yilmaz, N.K, Evangelinos, C., Lermusiaux, P., & Patrikalakis, N.M. (2008). Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming. *IEEE Journal of Oceanic Engineering*, *33*(4), 522–537.

Yoon, S.W., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling* (pp. 352-359). Providence, RI: AAAI Press.

Young, J., & Hawes, N. (2012). Evolutionary learning of goal priorities in a real-time strategy game. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Stanford, CA: AAAI Press.

Challenges in Finding Ways to Get the Job Done

Iman Awaad and **Gerhard K. Kraetzschmar**
Bonn-Rhein-Sieg University
and B-IT Center
Grantham-Allee 20
53757 Sankt Augustin, Germany

Joachim Hertzberg
Osnabrück University
and DFKI RIC Osnabrück Branch
Albrechtstrasse 28
49076 Osnabrück, Germany

Abstract

Humans exhibit flexible and robust behavior in achieving their goals. We make suitable substitutions for objects, actions, or tools to get the job done. When opportunities that would allow us to reach our goals with less effort arise, we often take advantage of them. Robots are not nearly as robust in handling such situations. Enabling a domestic service robot to find ways to get a job done by making substitutions is the goal of our work. In this paper, we highlight the challenges faced in our approach to combine Hierarchical Task Network planning, Description Logics, and the notions of affordances and conceptual similarity. We present open questions in modeling the necessary knowledge, creating planning problems, and enabling the system to handle cases where plan generation fails due to missing/unavailable objects.

Introduction

Generating plans that allow robots to act robustly and efficiently is complicated by the restrictions that govern the real-world environments in which they operate. Domains are usually finite but still very large, as they include all objects within the environment. Actions are durative and have non-deterministic outcomes. The environment is only partially observable, and what the agent senses may be inaccurate due to noise and the limitations of the sensors. The dynamic nature of the world implies that exogenous events frequently occur. Yet, researchers have enthusiastically worked on enabling robots to robustly plan and act (for that is the ultimate goal) in the real world.

In the domestic service domain, there is the added supposition that the robots will carry out their tasks in a socially-expected and -accepted manner. It is also assumed that they will do so sufficiently well out-of-the-box, but are also able to proactively pick up knowledge about new objects and places in the world, how to accomplish new tasks (or the same tasks differently), as well as preferences and social norms. For example, a robot such as Jenny (see Figure 1) may be expected to serve tea or water a plant out-of-the-box, but also to learn that there is a particular teacup that is my favorite, where it belongs in the kitchen, and that I like my tea served in it.

Humans are able to find fixes and adjust their plans, almost effortlessly. In fact, humans *expect* each other to find ways to get the job done, and are often less concerned with *how* others accomplish the task. This requires the ability to apply fixes to failures both during planning and execution, such as substituting objects. For example, we drink water in a mug instead of a glass, or water the plants with a tea kettle instead of a watering can. Humans find shortcuts and take advantage of opportunities. Enabling such behavior in artificial agents is highly desirable and the focus of this work.

Figure 1: Jenny, a Care-O-bot 3, getting the job done in a domestic service environment

While planning in real-world environments is in itself a complex process, handling failures during plan execution, or during the plan generation process itself, is equally difficult but just as necessary. To address the challenge of finding appropriate, socially-acceptable substitutes, we argue that the functional affordances (Hartson 2003) of objects, what objects are meant to be used for, should play a major role (see (Awaad, Kraetzschmar, and Hertzberg 2013b)). Affordances are "opportunities for action" (Gibson 1979). We adopt Norman's definition of *perceived affordances* which states that *opportunities for action* are "based on *the actors's goals, plans, values, beliefs and past experience*" (Norman 2002).

Similarity plays a role in choosing object substitutes. However, common similarity measures used in robotics may not yield the desirable results; e.g. instead of the color of an object the presence of a handle may be much more crucial. For measuring the conceptual similarity between original objects and possible substitutes, we use *Conceptual Spaces* (CS) (Gärdenfors 2004). CS provide a multi-dimensional

feature space where each axis represents a quality dimension, for example brightness, intensity, and hue. Points in a conceptual space represent objects, while regions represent concepts. CS can also combine quality dimensions to represent shapes, such as a 'handle'. The importance of particular dimensions for given tasks would provide us with a more robust measure of the suitability of a substitute.

Previous work (Awaad, Kraetzschmar, and Hertzberg 2013a) introduced our approach of identifying viable substitutes for objects by iteratively relaxing constraints in a structured way and lifting plans to use them. This is accomplished through three reasoning phases: the first phase generates a focused planning problem, the second phase expands the domain where necessary and re-plans to use the substitute, while the third and final reasoning phase uses affordances during plan execution. The domain is constrained at first to use instances from an object's class. It is expanded either due to a failure to generate a plan or during execution to include objects with the same functional affordances and high conceptual similarity. Further expansions of the domain would consider objects which have a high conceptual similarity only, and then those with the same functional affordances only. The approach is presented in (Awaad, Kraetzschmar, and Hertzberg 2014), of which this paper is an extended version.

Related work

Agents may fail to generate plans due to incomplete information or fail to execute them in dynamic environments. Such domains are often referred to as open-ended domains. Further complicating matters is the fact that real world domains, such as a domestic environment, are hard to model, even with correct, seemingly complete and up-to-date information (all of which are hard to come by when dealing with robots) and with non-deterministic outcomes of actions. In addition, the sheer size of these domains, can make the planning problems quite large, even for the simplest of goals.

Researchers have addressed the problem of planning in open-ended domains by developing approaches that are capable of handling the various possibilities. Given the task of making tea, for example, and not knowing whether the cup is clean or dirty, a contingent plan would foresee both these situations and include a solution for each case. Such solutions do not scale well though. Conformant planning and the use of probabilistic approaches also address the problems arising from uncertainty.

Decision-theoretic planning (Boutilier, Dean, and Hanks 1999) is perhaps the quintessential approach in addressing the issue of planning under uncertainty. At the heart of all planning approaches is the choice of which action, among alternatives, to take to achieve a goal. Preferences, or the maximization of a utility function which adheres to them, is the mechanism which guides the choice in the presence of uncertainty. Unlike the decision-theoretic approach, we can be seen as actually generating the alternative choices (and considering the various plans which accomplish the task with different substitutes). Our approach makes no explicit mention or use of utility functions such as that presented in (Haddawy et al. 1993), however preferences play a major role in the decision making process. Preferences in how flexible the system may be in making substitutions dictates whether the agent is risk-averse (little flexibility), risk-neutral or risk-taking. The choice of how important a possible substitute's proximity is to the agent is the main mechanism that effects this behavior. If a user attaches a high weight to proximity, then an agent given two possible substitutes, one better than the other but farther away, would choose to use the closer one regardless of optimality. The structured relaxation of constraints by the system can be seen as the implementation of a policy determining how the choice of possible substitutes is to be made.

Intuitively, by tightly interleaving the planning and execution processes to handle the dynamic nature of the environment, a more robust outcome is achieved (as the saying goes: "the only good action in a plan is the first one"). In (Off and Zhang 2012), the authors develop a control system which enables the agent to detect when it has insufficient knowledge and plan to acquire it through sensing actions. Our HTN domain also plans for sensing actions and then replans with the new current state. For example, this is what is done in a tidying-the-room scenario where the agent needs to detect objects, at various locations and check if they belong there before returning those that don't to their proper locations. Flexibility is addressed by the authors in (Levihn et al. 2013) by employing reconsideration (re-planning) and foresight (taking advantage of opportunities).

Least-commitment planning approaches (Weld 1994) are also extremely appealing in domains such as ours. Our work can be seen as relying on many of the same mechanisms. To maximize flexibility, we lift our methods and operators as first described in (Awaad, Kraetzschmar, and Hertzberg 2013a). The variable bindings can be seen as being "refined" to allow the plans to use the substitutes by iteratively decreasing constraints in a structured way. Moreover, the substitution of objects may result in the need to add or remove operators, e.g. filling a kettle requires a lid to be opened, whereas filling a watering can does not. This can be seen as a refinement operation where steps are added or removed. Not only does this allow the agent to generate plans where it might otherwise fail to do so, it also enables more flexible execution as any instance of these objects can be used. In our case though, the refinement operations, including lifting, are accomplished outside of the planner itself through the modification of the domain before the problem is given to the planner.

While (Cox and Zhang 2007) focuses on improving the performance of novice users in mixed-initiative approaches, the authors address the case where planners fail due to insufficient resources or because of changes in the environment, as we do here. Their system allows human users to transform the goals directly via an interface, thereby steering the planning process. This is perhaps the most similar work to ours in that they, like us, can be seen as transforming goals to assign resources. Human users play a vital role in our approach too. First by setting the importance of proximity, which varies the obedience/flexibility of the system, and second by providing feedback on whether a substitution is acceptable or not, and possibly the reason why it is not,

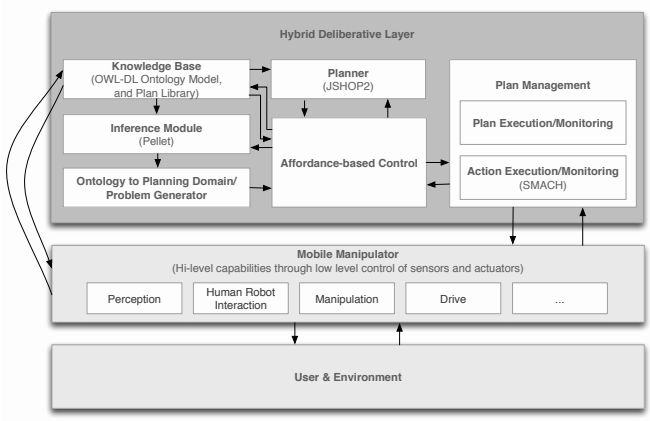


Figure 2: Software architecture for Jenny, extending the hybrid deliberative layer to use affordance-based reasoning in a domestic environment (Awaad, Kraetzschmar, and Hertzberg 2013a)

although the incorporation of this sort of knowledge into the system is still future work and beyond our current scope. This enables the system to improve its performance over time and according to user profiles. (Scala 2013) looked at assigning resources to enable robust execution of plans in real-world domains – addressing a similar problem to ours.

Others have investigated combined HTN and DL systems, e.g. (Sánchez-Ruiz, González-Calero, and Díaz-Agudo 2007; Sirin 2006; Gil 2005). Much work has gone into improving the efficiency of DL reasoning approaches for dynamic updates to the ABox (e.g. (Halashek-Wiener, Parsia, and Sirin 2006)). This is crucial when we consider the need to frequently change the instances in the ABoxes, as is necessary when updating them to reflect the changing state of the world.

Planning and acting on Jenny

An overview of our efforts to enable Jenny to accomplish tasks within the domestic service robot domain is given in this section. The proposed software control architecture towards which we are working is depicted in Figure 2. As a running robot system is needed at all times, stepwise development and integration processes have been undertaken to allow Jenny to plan and act. The planning, execution and monitoring processes described in this section are already integrated, have been tested in simulation, and are currently being tested on the real robot. The modules that enable the substitution of objects (KB, inference module, ontology to planning domain/problem generator and control) are under continuous development and evaluation apart from this integrated system and are presented in Section Approach.

The robot platform used is a Care-O-bot 3 robot (Graf et al. 2009), an omni-wheeled platform with a 7-degrees-of-freedom manipulator and a three-fingered gripper, running ROS (Quigley et al. 2009).

Procedural knowledge of *how* to accomplish tasks is encoded in Hierarchical Task Network (HTN) methods and operators (Erol, Hendler, and Nau 1994). In large domains, this approach allows us to compactly represent planning domains by directly encoding expert knowledge about carrying out a task. This results in high-quality plans.

The system maintains the state of the world, used for task planning, within the JSHOP2 (Ilghami and Nau 2003) problem file. The planning domain also includes the methods and operators necessary to decompose a number of real-world tasks, many of which tackle the incomplete information the system must deal with. For example, one such task is that of tidying up a room, where the robot performs sensing actions at various locations to identify objects which do not belong there and which should be returned to their given place, and then proceeds to do so. Such continual planning is often necessary in the domestic service robot domain.

The individual actions of a generated plan are executed by state machine-like execution scripts. These scripts are implemented in SMACH (Field 2011). Executed actions are monitored by the corresponding SMACH states (one for each operator) to determine if they were successfully accomplished, or not. The feedback from this monitoring process signals the sequencer that the next action within a plan may be sent if the previous action was successfully executed and verified. Currently, actions are only executed sequentially and not in parallel. Otherwise, depending on the particular reason for failure and the particular actions, a number of retries may be attempted (e.g. for grasping actions) or re-planning may be triggered. The decision on how to proceed is encoded in the SMACH scripts by the designer, based on experience with such faults.

The system is able to continuously plan or re-plan, when necessary, due to the maintenance of the planner's problem file which constitutes the Knowledge Base (KB). Additional components update the KB based on executed actions. Specifically, the planner was extended to provide the grounded add and delete lists for the executed actions. This knowledge is used to automatically update the problem file to reflect the current state of the world during execution. In addition, an update is triggered when planned sensing actions are executed, for example, when Jenny needs to check how many people need to be served, or when checking for objects which may need to be tidied up.

Additional extensions provide traces of the plan generation process. The main goal of these extensions is to provide information about the specific causes for failing to generate a plan for a given problem. This allows a decision to be made on whether a substitution is to be attempted or not. The trace outputs the task network, precondition(s) which resulted in the failed planning process and the bindings (or lack thereof for the given variables). How this is used to find and make substitutions is presented in the upcoming sections. The traces have had the added benefit of helping the modeler to optimize the domain, in particular by using the information on backtracking during the plan generation process.

Problems and challenges

A number of hurdles need to be overcome to enable Jenny to robustly plan and act in the real world. Many of these

hurdles arise from the limitations of specific integrated components and the lack of certain functionalities. We assume, for example, that the perception capabilities of the robot allow it to detect objects, determine if they are clean or dirty and, when attempting a substitution, measure the similarity between two objects. That said, such perception tasks are in fact not at the same level of development. Detecting whether objects are clean or dirty, for example, is not currently possible in our system.

Other challenges are due to the perceptual anchoring problem: "connecting, inside an artificial system, symbols and sensor data that refer to the same physical objects in the external world" (Coradeschi and Saffiotti 2003). Updating the KB based on sensing actions is complicated by the uncertainty in identifying identical instances of objects. For example, a bottle which is observed to be standing on a table may be added to the KB before the agent proceeds to carry out tasks elsewhere. Upon sensing the bottle again on the table, possibly at a different position, and given that only a part of the table may have been analyzed in the initial scene, the decision of whether the perceived bottle is the same instance of the bottle, which needs to updated with respect to its location, or a new instance, which needs to be added to the KB, becomes a complicated problem.

Similarly, while multiple instances of objects have unique identifiers (labeled by the perception system), and may be added to the KB, it is virtually impossible to identify which instance actually corresponds to the one used in the plan unless the object was being tracked continuously. For example, two teacups may have been detected on a table leading four predicates to be added to the KB: (`teacup teacup7`), (`teacup teacup8`), (`on table3 teacup7`) and (`on table3 teacup8`). The planner would use the first instance that satisfies all the preconditions to ground the operators. When executing the action however, as the perception system's ability to distinguish one instance from another is lacking and as there is no integrated KB which would include the global poses of the detected objects, either teacup may end up being used. While there has been much research on modeling knowledge in robotic systems, e.g. (Elfring et al. 2013), a KB that links the knowledge used by the various robot components continues to be a major research focus for our group.

Such scenarios become even more complicated in a continuous planning setting. For example, during the execution of sensing actions that have been planned to identify whether the cups are clean or dirty, or empty or full (given that such functionality exists), distinguishing between instances and updating the KB accordingly becomes very problematic. The agent can not know which of the two identical teacups on the same table to associate the sensed property with.

Incomplete information is the status quo in domains such as ours. The lack of information to generate a plan is a common reason for plan generation failures. For some tasks, like that of tidying up the room, the continuous planning approach works well. However, consider the case where the robot is in the living room and is asked to make a cup of tea. Not knowing whether there is milk in the fridge or not (an assertion (`have milk`), needed as a precondition by an operator, may be missing from the description) may prevent a plan from being generated. Including such details as everything within the fridge (and each cupboard and drawer in each room) has the downside of blowing up the domain, as well as making it more difficult to maintain a consistent and updated KB. This is one of the motivations for constraining the domain in our approach to include only relevant domain information for a given problem (see next section).

A clever designer will model the domain in such a way as to plan for the most-likely scenario and enable the system to handle contingencies. This is precisely what we attempt to do in our approach and is the main motivation for our work.

Robust monitoring of executed actions poses its own challenge due to the non-deterministic nature of the action outcomes. Currently, the cheapest monitoring action is used. This may not be as robust as necessary. For example, when releasing a bottle on a table, the cheapest action is to check the tactile sensors in the robot's fingers to see if the object is no longer being grasped. This monitoring action, however, says nothing about whether the bottle is standing on the table as expected, was knocked over, or if it has fallen off the table altogether. More expensive actions would need to be taken in order to recognize such situations. Similarly, it is not always feasible to confirm that all the preconditions of the following action within the plan are met. In addition, executing monitoring actions may lead to cycles or unacceptably long execution times. For example, in order to monitor the outcome of an action that releases a bottle on the table through a vision action (instead of the tactile sensors), Jenny would need to move the arm out of view of the camera. This additional action would itself need to be monitored. The robot would then perform the original monitoring by detecting the object on the table before once again having to return the arm to its position which again would trigger a monitoring action to confirm that it has reached its position.

Executing actions in parallel poses many additional difficulties. For example, moving to a location *while* searching for a given object is a behavior that is currently implemented as a sequential plan (moving along the path and then searching before moving again, and so on). Many more challenges of this or similar kind exist.

Approach to getting the job done

In this section, we provide the details of our approach and show how we extend the current implementation, described above, to allow the substitution of objects to get the job done. The work described in this section is partially reproduced from (Awaad, Kraetzschmar, and Hertzberg 2014).

The first step involves modeling the domain such that it supports the creation of a constrained planning problem: a step often provided manually in robotics as the initial state of the world is difficult to ascertain. This is also necessary due to the large size of the complete real-world domain as there is no way to automatically identify the relevant information to include for a particular planning problem.

The explanations of why a plan generation process failed are provided by the planner, through the extensions described above, and are used by the control module (as seen in Figure 2) to help determine if a substitution should be

attempted. This also enables the module to expand the domain to include substitutable objects which should allow the robot to get the job done – i.e. to include instances of the same class, those with the same functional affordances and/or those which are conceptually similar. Two illustrative examples are also presented to exemplify the approach from modeling to execution.

We model the domain, including functional affordances and parts of objects, in Description Logics (DL). This allows us to leverage the power of existing tools, such as the Pellet reasoner (Sirin et al. 2007), to infer implicit information from the explicitly modeled knowledge. It also enables us to identify situations where we don't have the necessary information (as it operates under the open world assumption). The value of integrating HTN planning and DL has been shown (empirically) in (Hartanto 2011). The methods and operators are transformed into the OWL-DL syntax as proposed in (Hartanto 2011) to allow us to use more domain knowledge to assert a focused planning problem.

In cases where planning fails due to a missing object, the algorithm reasons about possible substitutes and expands the domain to include them. Such a choice may be the most appropriate substitution, or the cheapest one due to its spatial proximity, or some weighted combination of both. The agent, like humans in most situations, displays bounded rationality: proposing a satisfactory solution, given the limited resources (within the environment as opposed to cognitive resources), as opposed to the optimum one specified in the task networks. We believe that this approach is key to our own flexibility.

Modeling the domain

Two tasks are used to demonstrate how the domain is modeled. The first task involves the robot making a cup of tea and the second involves it watering a plant. The knowledge base (KB) holds all models for the domains. The methods and operators that decompose the tasks are transformed into the OWL 2 DL format (cf. (Hartanto 2011)) and are included in the TBox. The conceptual knowledge of the world is also saved there. The ABox holds the knowledge specifying the state of the world as it changes as well as the constrained planning domain. When a new task is assigned, this information is stored in the KB. When new objects are perceived, or when actions change the state of the world, this too is reflected in the KB.

The KB provides a common knowledge source for agents and their components. The concepts of objects and locations which are used by the motion and grasp planners originate within this KB, although these planners also use additional KBs (e.g. OCL in the case of the grasp planner). Providing task-relevant information to the motion and grasp planners is also part of the domain model.

In addition to the KB, a blackboard is used to communicate lower-level information, in particular, it is where affordance cues (Fritz et al. 2006), in the form of CS quality dimensions, are posted as the agent moves through its environment while executing a plan. These CS might be of varying complexity (from simple color hues which would cost very little in terms of perceptual processing to more complex

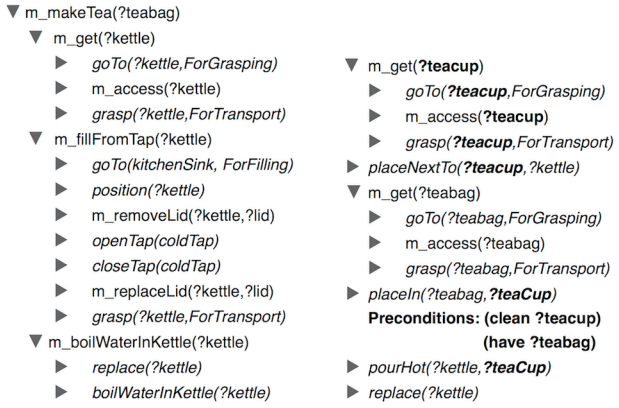


Figure 3: Sample decomposition of the tea-making task (Awaad, Kraetzschmar, and Hertzberg 2014)

concepts such as shape which might have been picked up as part of the plan's execution) and would be kept in the system for a given duration. Upon plan failure, the cues which are in close proximity can be used to identify viable candidates for substitutions. The same cues allow the agent to take advantage of opportunities as it carries out tasks during execution. This is what (Levihn et al. 2013) call "serendipity" in a navigation domain. Exploiting this in our system is currently future work, but it is another motivating factor for a distinct affordance-based approach. In situations where little is truly within the agent's control, it makes sense to consider what opportunities for action the environment affords, rather than considering those which it may or may not provide. For example, a cupboard full of glasses would guide the agent to grasp any of them.

The tea-making task

A sample decomposition specified by methods and operators in the domain for the tea-making task is given in Figure 3. It calls for a clean teacup to be used. Substituting objects, in the case a clean teacup cannot be found, is in essence allowing other object types to bind with the ?`teacup` variable. The variable types used in the planning domain are modeled as classes in the ontology. The ontology itself builds on the upper ontology of (Tenorth and Beetz 2009) and as such includes concepts such as `SpatialThing`, `TemporalThing`, `AgentGeneric` and so on. Another part of the ontology models those concepts necessary when converting between OWL and JSHOP syntaxes. The rest of the ontology is modeled based on the dictionary definition of objects. No *ad hoc* categories are created as is often the case in other work described in the literature.

Dictionary definitions aim to be concise and unambiguous, providing information that can help perception components to 'search' for the relevant cues and allow them to trigger afforded behaviors when sensed. In addition, they also tend to provide the functional affordances of objects. For example, a teacup, is defined as "a cup from which tea is drunk" (McKean 2005), and a cup is "a small, bowl-shaped container for drinking from, typically having a han-

dle" (McKean 2005). Therefore, dictionaries make good sources to guide the modeling of the ontology.

The models for a cup and a teacup, for example, are given here in Manchester OWL Syntax:

```
Class : Cup
SubClassOf : Container
AND (hasObjectToActOn some Liquid)
AND (hasShape some BowlShaped)
AND (isUsedFor some DrinkingFrom)
AND (hasSize only Small)
AND (canHavePart some ObjectPart)

Class : Teacup
EquivalentTo : Cup
AND (hasObjectToActOn some Tea)
AND (isUsedFor some DrinkingFrom)
```

Functional affordances are modeled by the `isUsedFor` property. As a *defined class*, any instance that is a member of the `Cup` class and that is used to drink tea from will be inferred by the reasoner to be a `Teacup`. These are necessary and sufficient conditions that enable the KB to be used for inference and not simply as a database. In addition to these properties, the `Teacup` concept also inherits various other properties from the superclass concepts (including additional `isUsedFor` properties). In this case, it inherits everything from the `Cup` and `Container` superclasses and so inherits everything described above for `Cup`, and two more properties inherited from `Container`: `isUsedFor some : Holding` and `isUsedFor : some : Transporting`.

`Holding`, `Transporting`, and `DrinkingFrom` are all examples of the `ActionOnObject` concept. This is the superclass of all the functional affordances of objects and parts of objects.

The `isAlsoUsedFor` property enables new functional affordances to be learned through experience, for example, by having been successfully substituted for a task. A bottle is defined as "a container, typically made of glass or plastic and with a narrow neck, used for storing drinks or other liquids" (McKean 2005). Should a user drink from the bottle, the property `isAlsoUsedFor some DrinkingFrom` would be added to the `Bottle` concept. This would also provide a quick way to look up which objects have previously been approved as substitutes and thus, allows the transfer and re-use of this knowledge.

We extend the functional view of objects to our actions by modeling the reason why the action is being performed so that the motion and grasp planners can, for example, provide suitable poses. The operator `goTo(?kettle, ForGrasping)` shown in Figure 3, for example, communicates the reason why the agent needs to go to the kettle. This information is used by the motion planner to take into consideration the necessary constraints in identifying the final pose to reach. These constraints may also be influenced by the perception components (it needs to be in a position that would allow it to localize the kettle in order to proceed with the grasping action). In addition, a constraint-based system for grasping (Schneider 2013) uses the information to verify that the action can indeed be performed using the specified object and with the specified hardware. For example, the action grasp(?cleanerBottle,ToSpray) would trigger the system to validate that such a task is possible. This is important as faults can occur at any time and it is therefore not sufficient to specify capabilities once. Moreover, these capabilities depend on the agent, the object and the intended use, hence the use of affordances to link them.

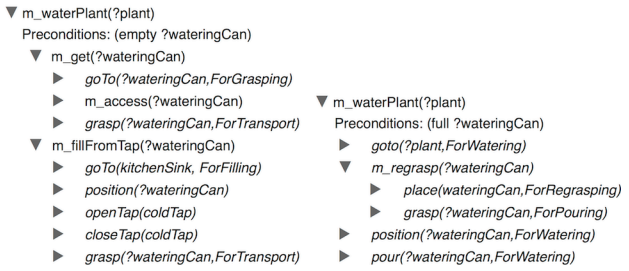


Figure 4: Sample decomposition of the plant-watering task (Awaad, Kraetzschmar, and Hertzberg 2014)

The plant-watering domain

This scenario serves to highlight the limitation of using functional affordances to make substitutions. When objects have very specific uses, it may no longer be possible to find other objects that share the same affordance. The opposite is also true: When many objects share the same functional affordance, they would all be equally likely to be used as substitutes for each other, although there may be some which would make better substitutes. For example, a glass, teacup, cup, mug, and bottle may all share the `DrinkingFrom` concept. However, the best substitute object is very often the one most similar to the original. In both these cases, a means to measure the conceptual similarity between objects would allow the better choice to be made. This example also demonstrates the fact that, in some situations, the plans themselves may need to be adapted, and why goal transformation and replanning alone are insufficient.

The domain specification for the plant-watering task is straightforward. Figure 4 shows a sample task decomposition specifying that a watering can should be fetched, filled, and used to water a plant. For this scenario, let us assume that a watering can is not available and so, to avoid failing to accomplish the task, a substitution is attempted. Querying for another object which `isUsedFor Watering Plants` yields no instances, and neither does a query for `isUsedFor Watering` alone.

As mentioned above, the similarity measures which are often used may not yield the results we have in mind. Hence our use of CS. Learning the relation between these quality dimensions and given tasks would allow the agent to choose the most appropriate object within the KB. For example, for watering plants, the capacity to hold water is perhaps the most important concept, followed by the presence of a handle and a spout, and in this case, the relevant query may return the tea kettle. Such relations could then be used as

weighting factors to determine how well an object would substitute for another in achieving a given task (similarity is measured as the weighted Euclidean distance). The modeling of the objects in CS and learning these weights is actively being investigated.

The need to transform the plans as well can be seen in the case a tea kettle is substituted for the watering can. Two additional actions are necessary to remove the kettle's lid, and replace it during filling (as seen in Figure 3). These are absent from the `fill` method for the watering can seen in Figure 4. Once again, the solution lies in the modeling of the methods and operators in DL.

Generating the planning problem

Having shown how we model the domain, we now look into how to generate a constrained planning problem. By matching a user's goal to a task, we are able to query our KB for all methods and operators which could decompose the given task. This is one of the benefits gained from using a hierarchical approach.

The approach used in (Hartanto 2011) explicitly specifies, for each method and operator, a `useState` variable which contains a list of states to be included within the initial state. For example, a navigation domain would include only rooms with open doors in the initial state. This additional domain knowledge is what enables the generation of a constrained planning problem.

The planner then attempts to generate a plan and, in the best-case scenario, succeeds. Unfortunately, things can, and often do, go wrong. The most likely scenario is that a method or operator requires that a precondition be met and the KB lacks the information to determine if it is or isn't. For example, it contains no information on whether or not there exists an instance of a clean cup, even if it knows that all known instances are dirty.

Humans faced with the same situation would still attempt to proceed with their plan and assume that they will adapt as need be. As long as there are instances of the object, we attempt to generate a plan. If the locations of the instances are unknown, this flexibility is achieved by querying a semantic map and creating dummy instances as needed at the most probable location. For example, a dummy instance of `clean`(`cup`) may be instantiated `in`(`cupboard4`) in the ABox. This allows the planning process to proceed and increases the chances of successfully getting the job done.

Assuming we have sufficient information within our KB to determine if preconditions are met or not, and even with the help of these dummy instances, the planner may still fail to generate a plan due to a failed precondition. For example, all cups may in fact be dirty. In such a case, humans would consider either washing a cup or making a substitution that would allow them to get the job done. They may use a mug, for instance. Here, we assume that, if a method allowing the robot to wash a cup while making tea was desirable to the user, it would have been included in the tea-making method decomposition and `dirty`(`teacup`) would be the filter condition that would allow that branch to be taken. However, in our domain, washing the cup is undesirable (because our robot simply can't wash the cup and the dishwasher would take too long) and so the agent has no choice but to attempt a substitution.

If we recall how our planning problem was generated, the initial state is constrained and only contains `Teacup` instances, as that is what the methods and operators specify. The domain needs to be expanded to include the next best possible substitute for a teacup. Then, a means to enable the new object to bind to the variables in the existing methods and operators needs to be found. This is necessary to handle cases such as the one described above where filling a kettle to water plants involves additional actions which are missing from the original plant-watering domain.

Expanding the domain

The expansion process can be seen as climbing a "flexibility ladder" (Awaad, Kraetzschmar, and Hertzberg 2013a) where constraints are iteratively decreased in a structured way to provide the flexibility with which to choose substitute objects. Substitutions are attempted when the plan generation process has failed. The planner outputs explanations which include a reason for the failure, such as the failed preconditions, the bindings which have been made, as well as the task network up to the point of failure.

The control module is responsible for providing the guidelines to expand the domain. To do this, it combines functional affordances and conceptual similarity to create the appropriate queries for possible substitutes. If the goal specified the use of a unique instance (e.g. my teacup), the domain is expanded to include other instances of the class which the given object belongs to, for example, instances of a teacup would be included in the new problem's initial state. The methods and operators are adapted to use the new object type.

If it fails to find such instances and the creation of a dummy instance is not supported by the semantic map (due to low probabilities), then the domain is expanded to include instances of objects which satisfy the next set of constraints: objects with the same functional affordance as the original object and high conceptual similarity, for example a mug. This seems to be in line with our own preferences.

The next higher level would remove the constraint that the substitute should be conceptually similar, relying only on a shared functional affordance, for example, a vacuum flask. Should the agent not find such objects and given the old adage that "form follows function" (the form of objects is based on their function), conceptual similarity is then used to identify those objects which do not share the same functional affordance and yet are conceptually similar, for example a measuring cup; or in the case of the watering can, a tea kettle.

As previously mentioned, a direct substitution of objects may not produce a sound plan. Therefore, in addition to the objects being included in the newly expanded domain, the methods and operators which use them are also included. For example, the `m_fillFromTap`(?`kettle`) method with its additional steps replaces the original `m_fillFromTap`(?`wateringCan`) method.

To accomplish this, a successful decomposition of the original task is needed. The domain expansion process there-

fore involves two stages. In the first stage, the failed preconditions/bindings are 'corrected' (e.g. by using a placeholder instance of a clean `teacup`) and sent back to the planner so that it may successfully generate a plan. This process may be repeated until a plan is generated. Once a decomposition is available, method and operators that have the substitute object type as a variable are included in the newly expanded domain description along with instances of the substitute object. The preconditions (and variables) that referred to the original object, e.g. `have(?teacup)`, are replaced with the substitute, e.g. `have(?mug)`, and the planning process is triggered once more to enable the substituted object to be used in the new plan.

Challenges in getting the job done

Plan-space planning was considered, as an alternative to HTN planning, since the plan refinement process lends itself to the refinement of partial plans to use the substituted objects. Unfortunately, the system would lose the benefits of using the hierarchical approach; not least of which is the ability to identify and include in the problem description only those methods and operators which are relevant to achieving a given task and the relevant states specified within them in order to create the focused problem.

We have so far argued that objects (and actions) may be substituted to allow agents to get the job done. Yet, the fact remains that not all objects may be substituted. For example, the key to the house may not be substituted by other instances of a key. The same holds for locations – if the task is to clean the dining room, and it is inaccessible, another room should not be substituted. Identifying such cases automatically is a challenge. To help overcome this, and for safety reasons, we rely on the user to confirm or oppose the choice of the recommended substitute, and possibly make his/her own recommendation.

Modeling the domain poses numerous challenges. There is little in the way of documentation or best practices on domain modeling. One issue is how to model transient properties of objects. For example, a common teacup may become "Fred's teacup" for the duration he is using it and then revert once more to its original status. Similarly, representing that a cold cup of coffee no longer has the functional affordance of drinking is tricky to model in a solid and general way. This is not to be confused with situations where the functional affordances do not enable an action. For example, while a cup is for drinking from, a dirty cup or one that is currently being used, may not offer this affordance. The same goes for a closed door. Such situations are handled in the filter preconditions of the method or operator that makes use of the affordance.

Modeling unique objects, such as my favorite teacup – of which only one could exist, is also slightly problematic as it needs to be handled as a class with only one instance. For the planner, typing the object would be the only way to specify that the unique instance should be used, however, this typing would need to be reflected in the methods and operators associated with making a cup of tea for a given user. Ways to specify such constraints in a more elegant manner are being investigated. One possibility would be to add a new decomposition as a branch to the method with a precondition specifying the user.

As mentioned above, constraining the domain to include relevant information for a given problem helps to maintain tractability, but the problems associated with maintaining a consistent and updated KB remain.

Highly-expressive query languages (QL), such as SPARQL-DL (Sirin and Parsia 2007), allow mixed TBox/RBox/ABox queries to be made. This is important as querying the ABox alone is not always sufficient. Differentiating between inherited properties and non-inherited properties might only be possible through mixed queries. For example, querying for objects with the property `usedFor Holding Liquids` returns instances of `Container` as well as all of its descendants. It may be desirable however to have such a query return only containers and not cups and teacups too. We are currently investigating whether such a QL would resolve this particular issue.

In (Hartanto 2011), an additional variable, `useState`, is specified for each method and operator in OWL-DL. It contains a list of states to be included within the initial state as part of the planning problem description. For example, a navigation domain would include only rooms with open doors in the initial state. This additional domain knowledge is what enables the generation of a constrained planning problem and provides the intensional representation of the world. It would be desirable to include this information directly within the preconditions and to simply query the KB for all preconditions in order to create the planning problem. This would keep to the existing JSHOP syntax and remove the need to specify additional information. Unfortunately, it would also lead to a tight coupling of the ontology to the domain, making the domain less general. In addition, not all preconditions can be queried (or queried with the expected answer) at the start of the planning process. For example, querying the KB to check if there are any objects at another location that don't belong during a tidying up task, is impossible without first driving there and performing the sensing actions. Checking whether an object is within the dextrous workspace of the robot before it has even approached it makes little sense, and would invariably return a useless answer at that stage. Therefore, this 'extension' makes little sense due to the complications arising from such fluents and the continual planning approach.

We assume the soundness provided by the planning approach, the planner and the consistency of the planning domain. Therefore, one possible complication is that the consistency of the planning domain may be compromised with the transformations made to the methods and operators to use a substituted object. We rely on user feedback to avert problems that arise due to this issue.

To the best of our knowledge, no existing system addresses the problem we do and a comparative study is therefore not possible. Flexibility and robustness are rather difficult to measure empirically. While our approach would provide a way to get the job done where other approaches simply do not, the design of a meaningful and thorough evaluation of the system remains a challenge.

Conclusions and future work

The goal of our work is to identify and use a substitute for an object which is needed to generate or execute a plan, but is either unknown during planning or cannot be found during execution, and thereby leads to a failure. This paper presented the approach, demonstrated how it addresses the problem, and motivated and explained the various design decisions. The mechanisms described also allow the agent to behave opportunistically.

The OWL-DL ontology includes the objects, parts and functional affordances which allow us to evaluate the system. The investigation of the possibility to autonomously, or semi-autonomously, acquire these objects and functional affordances from online sources would be useful, but is outside the scope of this work.

In addition to our work on substituting objects, we have also investigated action substitution in the form of affordance-based action abstraction (Höller 2013) where the agent learns which actions (as opposed to objects, as discussed in this paper) may be substituted and executed successfully in a given case. The preconditions and effects are learned and represented in a (CS) framework. Behaviors with similar effects on an object are clustered together to form abstract operators which are used during the planning process. During execution, the context (i.e. the state before an action is executed) is compared to previous execution runs (previous contexts) and the behavior which is predicted to have the highest success rate is instantiated and performed. In the case of failures at execution time, the actions are substituted by instances from the same cluster (those with the next-highest success prediction). This work was evaluated in the OpenRAVE (Diankov 2010) simulator. While we have based our formalization of conceptual spaces as vector spaces to measure the similarity of actions, contexts, and outcomes on that of (Raubal 2004), we have not yet addressed the measuring of similarity between two objects.

We continue to extend our domestic service planning domain to include more tasks. We are in the process of migrating from using SMACH to ROS Decision Making for a more robust execution and monitoring framework.

We are deriving metamodels of JSHOP2 in OWL-DL and in Ecore (The Eclipse Foundation 2013) which allow us to perform model-to-model transformations between these two representations. Model-to-model transformations would allow us to either model the planning domain in the JSHOP2 representation (i.e. JSHOP2 grammar) and then automatically generate the corresponding OWL-DL representation or vice versa. Additionally, the model-driven approach allows us to cleanly separate the JSHOP2 domain models and the model-to-model transformations from their implementation in a general-purpose programming language such as Java or C++. This implementation would be automatically generated.

In our ongoing research, we are continuing to investigate the representation of objects in CS and the specification, then learning, of the objects' quality dimensions' weights for a given task. We are also looking into the means to manage profiles and preferences.

Acknowledgments.

Iman Awaad gratefully acknowledges financial support provided by a PhD scholarship from the Graduate Institute of Bonn-Rhein-Sieg University. The authors thank Elizaveta Shpieva, Daniel Höller, Christian Tiefenau and Sven Schneider for their help in implementing some of the ideas presented here. The authors thank Sven Schneider and the reviewers for their valuable feedback.

References

Awaad, I.; Kraetzschmar, G. K.; and Hertzberg, J. 2013a. Affordance-based reasoning in robot task planning. In *Planning and Robotics (PlanRob) Workshop at 23rd Intlernational Conference on Automated Planning and Scheduling (ICAPS)*.

Awaad, I.; Kraetzschmar, G. K.; and Hertzberg, J. 2013b. Socializing robots: The role of functional affordances. In *International Workshop on Developmental Social Robotics (DevSoR): Reasoning about Human, Perspective, Affordances and Effort for Socially Situated Robots at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Awaad, I.; Kraetzschmar, G. K.; and Hertzberg, J. 2014. Finding ways to get the job done: An affordance-based approach. In *Proccedings of the 24th International Conference on Planning and Scheduling (ICAPS)*.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Coradeschi, S., and Saffiotti, A. 2003. An introduction to the anchoring problem. *Robotics and Autonomous Systems* 43:85–96.

Cox, M. T., and Zhang, C. 2007. Mixed-initiative goal manipulation. *AI Magazine* 28(2):62–74.

Diankov, R. 2010. *Automated Construction of Robotic Manipulation Programs*. Ph.D. Dissertation, Carnegie Mellon University, Robotics Institute.

Elfring, J.; van den Dries, S.; van de Molengraft, M.; and Steinbuch, M. 2013. Semantic world modeling using probabilistic multiple hypothesis anchoring. *Robotics and Autonomous Systems* 61(2):95–105.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1123–1128. AAAI Press.

Field, T. 2011. SMACH documentation. Online at http://www.ros.org/wiki/smach/Documentation.

Fritz, G.; Paletta, L.; Dorffner, G.; Breithaupt, R.; and Rome, E. 2006. Learning predictive features in affordance based robotic perception systems. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 3642–3647.

Gärdenfors, P. 2004. How to Make the Semantic Web More Semantic. In *Proceedings of the Third International Conference (FOIS 2004)*, 17–34.

Gibson, J. J. 1979. *The ecological approach to visual perception*. Houghton Mifflin (Boston).

Gil, Y. 2005. Description logics and planning. *AI Magazine* 26(2):73–84.

Graf, B.; Reiser, U.; Hägele, M.; Mauz, K.; and Klein, P. 2009. Robotic Home Assistant Care-O-bot® 3 - Product Vision and Innovation Platform. In *Advanced Robotics and its Social Impacts (ARSO), 2009 IEEE Workshop on*, 139–144.

Haddawy, P.; Haddawy, P.; Hanks, S.; and Hanks, S. 1993. Utility models for goal-directed decision-theoretic planners. *Computational Intelligence* 14.

Halashek-Wiener, C.; Parsia, B.; and Sirin, E. 2006. Description logic reasoning with syntactic updates. In Meersman, R., and Tari, Z., eds., *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 722–737.

Hartanto, R., ed. 2011. *A Hybrid Deliberative Layer for Robotic Agents: Fusing DL Reasoning with HTN Planning in Autonomous Robots*. Berlin, Heidelberg: Springer-Verlag.

Hartson, H. R. 2003. Cognitive, physical, sensory, and functional affordances in interaction design. *Behaviour & IT* 22(5):315–338.

Höller, D. 2013. Affordance-based action abstraction in robot planning. Master's thesis, Bonn-Rhein-Sieg University of Applied Sciences.

Ilghami, O., and Nau, D. S. 2003. A General Approach to Synthesize Problem-Specific Planners. Technical Report CS-TR-4597, UMIACS-TR-2004-40, University of Maryland.

Levihn, M.; Kaelbling, L. P.; Lozano-Perez, T.; and Stilman, M. 2013. Foresight and reconsideration in hierarchical planning and execution. In *IEEE/RSJ - International Conference on Intelligent Robots and Systems (IROS): Workshop on Cognitive Assistive Systems*.

McKean, E., ed. 2005. *The New Oxford American Dictionary*. Oxford University Press.

Norman, D. 2002. *The psychology of everyday things*. Basic Books (New York).

Off, D., and Zhang, J. 2012. Continual HTN planning and acting in open-ended domains - considering knowledge acquisition opportunities. In *ICAART*, 16–25.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T. B.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Raubal, M. 2004. Formalizing conceptual spaces. In Varzi, A., and Vieu, L., eds., *Proceedings of the 3rd International Conference on Formal Ontology in Information Systems (FOIS 2004)*, 153–164.

Sánchez-Ruiz, A. A.; González-Calero, P. A.; and Díaz-Agudo, B. 2007. Planning with description logics and syntactic updates. In Salido, M. A., and Fdez-Olivares, J., eds., *Planning, Scheduling and Constraint Satisfaction (CAEPIA 2007 Workshop)*, 140–150. Universidad de Salamanca.

Scala, E. 2013. *Reconfiguration and Replanning for Robust Execution of Plans Involving Continuous and Consumable Resources*. Ph.D. Dissertation, Universita' degli Studi di Torino.

Schneider, S. 2013. Design of a declarative language for task-oriented grasping and tool-use with dextrous robotic hands. Master's thesis, Bonn-Rhein-Sieg University of Applied Sciences, St. Augustin, Germany.

Sirin, E., and Parsia, B. 2007. SPARQL-DL: SPARQL Query for OWL-DL. In *Proceedings of the Third International Workshop on OWL: Experiences and Directions (OWLED '07)*.

Sirin, E.; Parsia, B.; Grau, B. C.; Kalyanpur, A.; and Katz, Y. 2007. Pellet: A practical owl-dl reasoner. *Web Semant.* 5(2):51–53.

Sirin, E. 2006. *Combining Description Logic Reasoning with AI Planning for Composition of Web Services*. Ph.D. Dissertation, University of Maryland, College Park, Maryland.

Tenorth, M., and Beetz, M. 2009. KnowRob - knowledge processing for autonomous personal robots. In *Intelligent Robots and Systems (IROS), 2009 IEEE/RSJ International Conference on*, 4261–4266.

The Eclipse Foundation. 2013. Eclipse Modeling Framework Project Core. Online at http://www.eclipse.org/modeling/emf/?project=emf.

Weld, D. S. 1994. An introduction to least commitment planning. *AI Magazine*.

Integrating Probabilistic Graphical Models and Declarative Programming for Knowledge Representation and Reasoning in Robotics

Shiqi Zhang
Department of Computer Science
Texas Tech University, USA
shiqi.zhang6@gmail.com

Mohan Sridharan
Department of Computer Science
Texas Tech University, USA
mohan.sridharan@ttu.edu

Michael Gelfond
Department of Computer Science
Texas Tech University, USA
michael.gelfond@ttu.edu

Jeremy Wyatt
School of Computer Science
University of Birmingham, UK
jlw@cs.bham.ac.uk

Abstract

This paper describes an architecture that combines the complementary strengths of declarative programming and probabilistic graphical models to enable robots to represent, reason with, and learn from, qualitative and quantitative descriptions of uncertainty and knowledge. An action language is used for the low-level (LL) and high-level (HL) system descriptions in the architecture, and the definition of recorded histories in the HL is expanded to allow prioritized defaults. For any given goal, tentative plans created in the HL using default knowledge and commonsense reasoning are implemented in the LL using probabilistic algorithms, with the corresponding observations used to update the HL history. Tight coupling between the two levels enables automatic selection of relevant variables and generation of suitable action policies in the LL for each HL action, and supports reasoning with violation of defaults, noisy observations and unreliable actions in large and complex domains. The architecture is evaluated in simulation and on physical robots moving objects to specific places in indoor domains; the benefit on robots is a reduction in task execution time of 39% compared with a purely probabilistic, but still hierarchical, approach.

1 Introduction

Mobile robots deployed in complex domains frequently have incomplete domain knowledge, and receive far more raw data from sensors than is possible to process in real-time. The descriptions of knowledge and uncertainty obtained from different sources may complement or contradict each other, and may have different degrees of relevance to current or future tasks. Widespread use of robots thus poses fundamental knowledge representation and reasoning challenges—robots need to represent, learn from, and reason with, qualitative and quantitative descriptions of knowledge and uncertainty. Towards this objective, our architecture combines the knowledge representation and non-monotonic logical reasoning capabilities of declarative programming with the uncertainty modeling capabilities of probabilistic graphical models. The architecture consists of two tightly coupled levels and has the following key features:

1. An action language is used for the HL and LL system descriptions and the definition of recorded history is expanded in the HL to allow prioritized defaults.
2. For any assigned objective, tentative plans are created in the HL using default knowledge and commonsense reasoning, and implemented in the LL using probabilistic algorithms, with the corresponding observations adding suitable statements to the HL history.
3. For each HL action, abstraction and tight coupling between the LL and HL system descriptions enables automatic selection of relevant variables and generation of a suitable action policy in the LL.

In this paper, the HL domain representation is translated into an Answer Set Prolog (ASP) program, while the LL domain representation is translated into partially observable Markov decision processes (POMDPs). The novel contributions of the architecture, e.g., allowing histories with prioritized defaults, tight coupling between the two levels, and the resultant automatic selection of the relevant variables in the LL, support reasoning with violation of defaults, noisy observations and unreliable actions in large and complex domains. The architecture is grounded and evaluated in simulation and on physical robots moving desired objects to specific places in indoor domains.

2 Related Work

Probabilistic graphical models such as POMDPs have been used to represent knowledge and plan sensor input processing, navigation and interaction for robots (Hoey et al. 2010; Rosenthal and Veloso 2012). However, these formulations (by themselves) make it difficult to perform commonsense reasoning, e.g., default reasoning and non-monotonic logical reasoning, especially with information not directly relevant to tasks at hand. In parallel, research in classical planning has provided many algorithms for knowledge representation and logical reasoning (Ghallab, Nau, and Traverso 2004), but these algorithms require substantial prior knowledge about the domain, task and the set of actions. Many of these algorithms also do not support merging of new, un-

reliable information from sensors and humans with the current beliefs in a knowledge base. Answer Set Programming (ASP), a non-monotonic logic programming paradigm, is well-suited for representing and reasoning with commonsense knowledge (Baral 2003; Gelfond 2008). An international research community has been built around ASP, with applications such as reasoning in simulated robot housekeepers and for representing knowledge extracted from natural language human-robot interaction (Chen et al. 2012; Erdem, Aker, and Patoglu 2012). However, ASP does not support probabilistic analysis, whereas a lot of information available to robots is represented probabilistically to quantitatively model the uncertainty in sensor input processing and actuation in the real world.

Researchers have designed cognitive architectures (Laird, Newell, and Rosenbloom 1987; Langley and Choi 2006; Talamadupula et al. 2010), and developed algorithms that combine deterministic and probabilistic algorithms for task and motion planning on robots (Hanheide et al. 2011; Kaelbling and Lozano-Perez 2013). Recent work has also integrated ASP and POMDPs for non-monotonic logical inference and probabilistic planning on robots (Zhang, Sridharan, and Bao 2012). Some examples of principled algorithms developed to combine logical and probabilistic reasoning include probabilistic first-order logic (Halpern 2003), first-order relational POMDPs (Sanner and Kersting 2010), Markov logic network (Richardson and Domingos 2006), Bayesian logic (Milch et al. 2006), and a probabilistic extension to ASP (Baral, Gelfond, and Rushton 2009). However, algorithms based on first-order logic for probabilistically modeling uncertainty do not provide the desired expressiveness for capabilities such as default reasoning, e.g., it is not always possible to express uncertainty and degrees of belief quantitatively, and assigning high probability values to default knowledge may not make best use of such knowledge (see experimental results in Section 4.2). Other algorithms based on logic programming that support probabilistic reasoning do not support one or more of the desired capabilities: reasoning as in causal Bayesian networks; incremental addition of probabilistic information; reasoning with large probabilistic components; and dynamic addition of variables with different ranges; see (Baral, Gelfond, and Rushton 2009). The architecture described in this paper is a step towards achieving these capabilities. It exploits the complementary strengths of declarative programming and probabilistic graphical models to represent, reason with, and learn from qualitative and quantitative descriptions of knowledge and uncertainty, enabling robots to automatically plan sensing and actuation in larger domains than was possible before.

3 KRR Architecture

This section describes our architecture's HL and LL domain representations. The syntax, semantics and representation of the corresponding transition diagrams are described in an *action language* AL (Gelfond and Kahl 2014). Action languages are formal models of parts of natural language used for describing transition diagrams. AL has a sorted signature containing three *sorts*: `statics`, `fluents` and `actions`. Statics are domain properties whose truth values cannot be changed by actions, while fluents are properties whose truth values are changed by actions. Actions are defined as a set of elementary actions that can be executed in parallel. A domain property `p` or its negation $\neg p$ is a domain literal. AL allows three types of statements:

a **causes** l_{in} **if** $p_0,\ldots,p_m$ (Causal law)
l **if** $p_0,\ldots,p_m$ (State constraint)
impossible $a_0,\ldots,a_k$ **if** $p_0,\ldots,p_m$ (Executability condition)

where a is an action, l is a literal, l_{in} is a inertial fluent literal, and $p_0,\ldots,p_m$ are domain literals. The causal law states, for instance, that action a causes inertial fluent literal l_{in} if the literals $p_0,\ldots,p_m$ hold true. A collection of statements of AL forms a system/domain description.

As an illustrative example used throughout this paper, we will consider a robot that has to move objects to specific places in an indoor domain. The domain contains four specific places: *office*, *main_library*, *aux_library*, and *kitchen*, and a number of specific objects of the sorts: *textbook*, *printer* and *kitchenware*.

3.1 HL domain representation

The HL domain representation consists of a system description $\mathscr{D}_H$ and histories with defaults $\mathscr{H}$. $\mathscr{D}_H$ consists of a sorted signature and axioms used to describe the HL transition diagram τ_H. The sorted signature: $\Sigma_H = \langle \mathscr{O}, \mathscr{F}, \mathscr{P} \rangle$ is a tuple that defines the names of objects, functions, and predicates available for use in the HL. The sorts in our example are: `place`, `thing`, `robot`, and `object`; `object` and `robot` are subsorts of `thing`. Robots can move on their own, but objects cannot move on their own. The sort `object` has subsorts such as `textbook`, `printer` and `kitchenware`. The fluents of the domain are defined in terms of their arguments:

$$loc(thing, place) \qquad (1)$$
$$in\_hand(robot, object)$$

The first predicate states the location of a thing; and the second predicate states that a robot has an object.These two predicates are *inertial fluents* subject to the law of inertia, which can be changed by an action. The *actions* in this domain include:

$$move(robot, place) \qquad (2)$$
$$grasp(robot, object)$$
$$putdown(robot, object)$$

The dynamics of the domain are defined using the following causal laws:

$$move(robot, Pl) \textbf{ causes } loc(robot, Pl) \qquad (3)$$
$$grasp(robot, Ob) \textbf{ causes } in\_hand(robot, Ob)$$
$$putdown(robot, Ob) \textbf{ causes } \neg in\_hand(robot, Ob)$$

state constraints:

$$loc(Ob, Pl) \textbf{ if } loc(robot, Pl),\ in\_hand(robot, Ob) \qquad (4)$$
$$\neg loc(Th, Pl_1) \textbf{ if } loc(Th, Pl_2),\ Pl_1 \neq Pl_2$$

and executability conditions:

$$\begin{aligned}
&\textbf{impossible } move(robot,Pl) \textbf{ if } loc(robot,Pl) \qquad (5)\\
&\textbf{impossible } A_1,\ A_2,\ \textbf{if } A_1 \neq A_2.\\
&\textbf{impossible } grasp(robot,Ob) \textbf{ if } loc(robot,Pl1),\\
&\qquad loc(Ob,Pl2), Pl1 \neq Pl2\\
&\textbf{impossible } grasp(robot,Ob) \textbf{ if } in_hand(robot,Ob)\\
&\textbf{impossible } putdown(robot,Ob) \textbf{ if } \neg in_hand(robot,Ob)
\end{aligned}$$

The top part of Figure 1 shows some state transitions in the HL; nodes include a subset of fluents (robot's position) and actions are the arcs between nodes. Although $\mathscr{D}_H$ does not include the costs of executing actions, these are included in the LL (see Section 3.2).

Histories with defaults A recorded history of a dynamic domain is usually defined as a collection of records of the form $obs(fluent, boolean, step)$ and $hpd(action, step)$. The former states that a specific fluent was observed to be true or false at a given step of the domain's trajectory, and the latter states that a specific action happened (or was executed by the robot) at that step. In this paper, *we expand on this view by allowing histories to contain (possibly prioritized) defaults describing the values of fluents in their initial states*. A default $d(X)$ stating that *in the typical initial state elements of class c satisfying property b also have property p* is represented as:

$$d(X) = \begin{cases} default(d(X)) \\ head(d(X), p(X)) \\ body(d(X), c(X)) \\ body(d(X), b(X)) \end{cases} \qquad (6)$$

where the literal in the "head" of the default, e.g., $p(X)$ is true if all the literals in the "body" of the default, e.g., $b(X)$ and $c(X)$, hold true; see (Gelfond and Kahl 2014) for formal semantics of defaults. In this paper, we abbreviate $obs(f, true, 0)$ and $obs(f, false, 0)$ as $init(f, true)$ and $init(f, false)$ respectively.

Example 1 *[Example of defaults]*
Consider the following statements about the locations of textbooks in the initial state in our illustrative example. *Textbooks are typically in the main library. If a textbook is not there, it is in the auxiliary library. If a textbook is checked out, it can be found in the office.* These defaults can be represented as:

$$\begin{aligned}
&default(d_1(X))\\
&head(d_1(X), loc(X, main_library))\\
&body(d_1(X), textbook(X))
\end{aligned} \qquad (7)$$

$$\begin{aligned}
&default(d_2(X))\\
&head(d_2(X), loc(X, aux_library))\\
&body(d_2(X), textbook(X))\\
&body(d_2(X), \neg loc(X, main_library))
\end{aligned} \qquad (8)$$

$$\begin{aligned}
&default(d_3(X))\\
&head(d_3(X), loc(X, office))\\
&body(d_3(X), textbook(X))\\
&body(d_3(X), \neg loc(X, main_library))\\
&body(d_3(X), \neg loc(X, aux_library))
\end{aligned} \qquad (9)$$

A default such as "kitchenware are usually in the kitchen" may be represented in a similar manner. We first present multiple informal examples to illustrate reasoning with these defaults; Definition 3 (below) will formalize this reasoning. For textbook tb_1, history $\mathscr{H}_1$ containing the above statements should entail: $holds(loc(tb_1, main\_library), 0)$. A history $\mathscr{H}_2$ obtained from $\mathscr{H}_1$ by adding an observation: $init(loc(tb_1, main\_library), false)$ renders the first default inapplicable; hence $\mathscr{H}_2$ should entail: $holds(loc(tb_1, aux\_library), 0)$. A history $\mathscr{H}_3$ obtained from $\mathscr{H}_2$ by adding an observation: $init(loc(tb_1, aux\_library), false)$ entails: $holds(loc(tb_1, office), 0)$.

Consider history $\mathscr{H}_4$ obtained by adding observation: $obs(loc(tb_1, main\_library), false, 1)$ to $\mathscr{H}_1$. This observation should defeat the default d_1 in Equation 7 because if this default's conclusion were true in the initial state, it would also be true at step 1 (by inertia), which contradicts our observation. The book tb_1 is thus not in the main library initially. The second default will conclude that this book is initially in the auxiliary library—the inertia axiom will propagate this information and $\mathscr{H}_4$ will entail: $holds(loc(tb_1, aux\_library), 1)$.

The definition of entailment relation can now be given with respect to a fixed system description $\mathscr{D}_H$. We start with the notion of *a state of transition diagram τ_H of $\mathscr{D}_H$ compatible with a description $\mathscr{I}$ of the initial state of history $\mathscr{H}$*. We use the following terminology. We say that a set S of literals is *closed under a default d* if S contains the head of d whenever it contains all literals from the body of d and does not contain the literal contrary to d's head. S is *closed under a constraint* of $\mathscr{D}_H$ if S contains the constraint's head whenever it contains all literals from the constraint's body. Finally, we say that a set U of literals is the *closure of S* if $S \subseteq U$, U is closed under constraints of $\mathscr{D}_H$ and defaults of $\mathscr{H}$, and no proper subset of U satisfies these properties.

Definition 1 *[Compatible initial states]*
A state σ of τ_H is *compatible* with a description $\mathscr{I}$ of the initial state of history $\mathscr{H}$ if:

1. σ satisfies all observations of $\mathscr{I}$,
2. σ contains the closure of the union of statics of $\mathscr{D}_H$ and the set $\{f : init(f, true) \in \mathscr{I}\} \cup \{\neg f : init(f, false) \in \mathscr{I}\}$.

Let $\mathscr{I}_k$ be the description of the initial state of history $\mathscr{H}_k$. States in Example 1 compatible with $\mathscr{I}_1$, $\mathscr{I}_2$, $\mathscr{I}_3$ must then contain $\{loc(tb_1, main\_library)\}$, $\{loc(tb_1, aux\_library)\}$, and $\{loc(tb_1, office)\}$ respectively. There are multiple such states, which differ by the location of robot. Since $\mathscr{I}_1 = \mathscr{I}_4$ they have the same compatible states. Next, we define *models* of history $\mathscr{H}$, i.e., paths of the transition diagram τ_H of $\mathscr{D}_H$ compatible with $\mathscr{H}$.

Definition 2 *[Models]*
A path P of τ_H is a *model* of history $\mathscr{H}$ with description $\mathscr{I}$ of its initial state if there is a collection E of *init* statements such that:

1. If $init(f, true) \in E$ then $\neg f$ is the head of one of the defaults of $\mathscr{I}$. Similarly, for $init(f, false)$.

2. The initial state of P is compatible with the description: $\mathscr{I}_E = \mathscr{I} \cup E$.
3. Path P satisfies all observations in $\mathscr{H}$.
4. There is no collection E_0 of *init* statements which has less elements than E and satisfies the conditions above.

We will refer to E as an *explanation* of $\mathscr{H}$. Models of $\mathscr{H}_1$, $\mathscr{H}_2$, and $\mathscr{H}_3$ are paths consisting of initial states compatible with $\mathscr{I}_1$, $\mathscr{I}_2$, and $\mathscr{I}_3$—the corresponding explanations are empty. However, in the case of $\mathscr{H}_4$, the situation is different—the predicted location of tb_1 will be different from the observed one. The only explanation of this discrepancy is that tb_1 is an exception to the first default. Adding $E = \{init(loc(tb_1, main\_library), false)\}$ to $\mathscr{I}_4$ will resolve the problem.

Definition 3 *[Entailment and consistency]*

- Let $\mathscr{H}^n$ be a history of length n, f be a fluent, and $0 \leq i \leq n$ be a step of $\mathscr{H}^n$. We say that $\mathscr{H}^n$ *entails* a statement $Q = holds(f,i)$ ($\neg holds(f,i)$) if for every model P of $\mathscr{H}^n$, fluent literal f ($\neg f$) belongs to the ith state of P. We denote the entailment as $\mathscr{H}^n \models Q$.
- A history which has a model is said to be *consistent*.

It can be shown that histories from Example 1 are consistent and that our entailment captures the corresponding intuition.

Reasoning with HL domain representation The HL domain representation ($\mathscr{D}_H$ and $\mathscr{H}$) is translated into a program in CR-Prolog, which incorporates consistency restoring rules in ASP (Balduccini and Gelfond 2003; Gelfond and Kahl 2014); specifically, we use the knowledge representation language SPARC that expands CR-Prolog and provides explicit constructs to specify objects, relations, and their sorts (Balai, Gelfond, and Zhang 2013). ASP is a declarative language that can represent recursive definitions, defaults, causal relations, special forms of self-reference, and other language constructs that occur frequently in non-mathematical domains, and are difficult to express in classical logic formalisms (Baral 2003). ASP is based on the stable model semantics of logic programs, and builds on research in non-monotonic logics (Gelfond 2008). A CR-Prolog program is thus a collection of statements describing domain objects and relations between them. The ground literals in an *answer set* obtained by solving the program represent beliefs of an agent associated with the program[1]; program consequences are statements that are true in all such belief sets. Algorithms for computing the entailment relation of AL and related tasks such as planning and diagnostics are thus based on reducing these tasks to computing answer sets of programs in CR-Prolog. First, $\mathscr{D}_H$ and $\mathscr{H}$ are translated into an ASP program $\Pi(\mathscr{D}_H, \mathscr{H})$ consisting of direct translation of causal laws of $\mathscr{D}_H$, inertia axioms, closed world assumption for defined fluents, reality checks, records of observations, actions and defaults from $\mathscr{H}$, and special axioms for *init*:

$$holds(F,0) \leftarrow init(F,true) \qquad (10)$$
$$\neg holds(F,0) \leftarrow init(F,false)$$

[1] SPARC uses DLV (Leone et al. 2006) to generate answer sets.

In addition, every default of $\mathscr{I}$ is turned into an ASP rule:

$$holds(p(X),0) \leftarrow c(X), holds(b(X),0),\ not\ \neg holds(p(X),0) \qquad (11)$$

and a consistency-restoring rule:

$$\neg holds(p(X),0) \stackrel{+}{\leftarrow} c(X), holds(b(X),0) \qquad (12)$$

which states that to restore consistency of the program one may assume that the conclusion of the default is false. For more details about the translation, CR-rules and CR-Prolog, please see (Gelfond and Kahl 2014).

Proposition 1 *[Models and Answer Sets]*
A path $P = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_{n-1}, a_n \rangle$ of τ_H is a model of history $\mathscr{H}^n$ iff there is an answer set S of a program $\Pi(\mathscr{D}_H, \mathscr{H})$ such that:

1. *A fluent $f \in \sigma_i$ iff $holds(f,i) \in S$,*
2. *A fluent literal $\neg f \in \sigma_i$ iff $\neg holds(f,i) \in S$,*
3. *An action $e \in a_i$ iff $occurs(e,i) \in S$.*

The proposition reduces computation of models of $\mathscr{H}$ to computing answer sets of a CR-Prolog program. This proposition allows us to reduce the task of planning to computing answer sets of a program obtained from $\Pi(\mathscr{D}_H, \mathscr{H})$ by adding the definition of a goal, a constraint stating that the goal must be achieved, and a rule generating possible future actions of the robot.

3.2 LL domain representation

The LL system description $\mathscr{D}_L$ consists of a sorted signature and axioms that describe a transition diagram τ_L. The sorted signature Σ_L of action theory describing τ_L includes the sorts from signature Σ_H of HL with two additional sorts `room` and `cell`, which are subsorts of sort `place`. Their elements satisfy the static relation *part_of(cell, room)*. We also introduce the static *neighbor(cell, cell)* to describe neighborhood relation between cells. Fluents of Σ_L include those of Σ_H, an additional inertial fluent: *searched(cell, object)*—robot searched a cell for an object—and two defined fluents: *found(object, place)*—an object was found in a place—and *continue_search(room, object)*—the search for an object is continued in a room.

The actions of Σ_L include the HL actions that are viewed as being represented at a higher resolution, e.g., movement is possible to specific cells. The causal law describing the effect of *move* may be stated as:

$$move(robot,Y)\ \ \textbf{causes}\ \ \{loc(robot,Y) : neighbor(Y,X)\}$$
$$\textbf{if}\ \ loc(robot,X) \qquad (13)$$

where X, Y are cells. This causal law states that moving from a cell can cause the robot to be in one of the neighboring cells[2]. The LL includes an additional action *search* that enables robots to search for objects in cells; the corresponding

[2] This is a special case of a non-deterministic causal law defined in extensions of AL with non-boolean fluents, i.e., functions whose values can be elements of arbitrary finite domains.

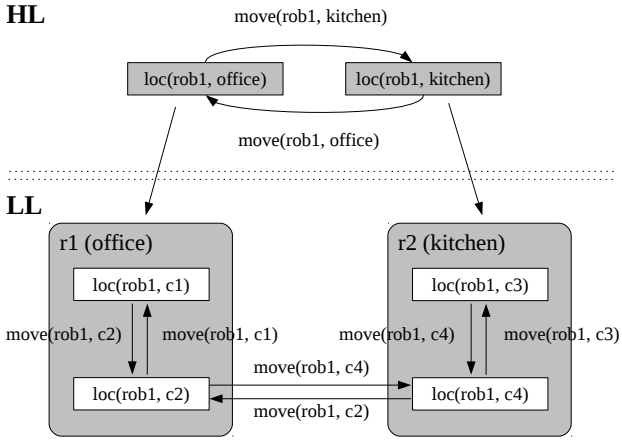


Figure 1: Illustrative example of state transitions in the HL and LL.

causal laws and constraints may be written as:

$$search(cell, object) \textbf{ causes } searched(cell, object) \quad (14)$$
$$found(object, cell) \textbf{ if } searched(cell, object), loc(object, cell)$$
$$found(object, room) \textbf{ if } part\_of(cell, room), found(object, cell)$$
$$continue\_search(room, object) \textbf{ if } \neg found(object, room), part\_of(cell, room), \neg searched(cell, object)$$

We also introduce a defined fluent *failure* that holds iff the object under consideration is not in the room that the robot is searching—this fluent is defined as:

$$failure(object, room) \textbf{ if } loc(robot, room), \neg continue\_search(room, object), \neg found(object, room) \quad (15)$$

This completes the action theory that describes τ_L. The states of τ_L can be viewed as extensions of states of τ_H by physically possible fluents and statics defined in the language of LL. Moreover, for every HL state-action-state transition $\langle \sigma, a, \sigma' \rangle$ and every LL state s compatible with σ (i.e., $\sigma \subset s$), there is a path in the LL from s to some state compatible with σ'.

Unlike the HL system description in which effects of actions and results of observations are always accurate, the action effects and observations in the LL are only known with some degree of probability. The state transition function $T : S \times A \times S' \rightarrow [0,1]$ defines the probabilities of state transitions in the LL. Due to perceptual limitations of the robot, only a subset of the fluents are observable in the LL; we denote this set of fluents by Z. Observations are elements of Z associated with a probability, and are obtained by processing sensor inputs using probabilistic algorithms. The observation function $O : S \times Z \rightarrow [0,1]$ defines the probability of observing specific observable fluents in specific states. Functions T and O are computed using prior knowledge, or by observing the effects of specific actions in specific states (see Section 4.1).

States are partially observable in the LL, and we introduce (and reason with) *belief states*, probability distributions over the set of states. Functions T and O describe a probabilistic transition diagram defined over belief states. The initial belief state is represented by B_0, and is updated iteratively using Bayesian inference:

$$B_{t+1}(s_{t+1}) \propto O(s_{t+1}, o_{t+1}) \sum_s T(s, a_{t+1}, s_{t+1}) \cdot B_t(s) \quad (16)$$

The LL system description includes a reward specification $R : S \times A \times S' \rightarrow \Re$ that encodes the relative cost or *value* of taking specific actions in specific states. Planning in the LL then involves computing a *policy* that maximizes the reward over a planning horizon. This policy maps belief states to actions: $\pi : B_t \mapsto a_{t+1}$. We use a point-based approximate algorithm to compute this policy (Ong et al. 2010). In our illustrative example, an LL policy computed for HL action *move* is guaranteed to succeed, and that the LL policy computed for HL action *grasp* considers three LL actions: *move*, *search*, and *grasp*. Plan execution in the LL corresponds to using the computed policy to repeatedly choose an action in the current belief state, and updating the belief state after executing that action and receiving an observation. We henceforth refer to this algorithm as "POMDP-1".

Unlike the HL, history in the LL representation consists of observations and actions over one time step; the current belief state is assumed to be the result of all information obtained in previous time steps (first-order Markov assumption). In this paper, the LL domain representation is translated automatically into POMDP models, i.e., specific data structures for representing the components of $\mathscr{D}_L$ (described above) such that existing solvers can be used to obtain action policies.

We observe that the coupling between the LL and the HL has some key consequences. First, for any HL action, the relevant LL variables are identified automatically, improving the computational efficiency of computing the LL policies. Second, if LL actions cause different fluents, these fluents are independent. Finally, although defined fluents are crucial in determining what needs to be communicated between the levels of the architecture, they themselves need not be communicated.

3.3 Control loop

Algorithm 1 describes the architecture's control loop[3]. First, the LL observations obtained in the current location add statements to the HL history, and the HL initial state (s^H_{init}) is communicated to the LL (line 1). The assigned task determines the HL goal state (s^H_{goal}) for planning (line 2). Planning in the HL provides a sequence of actions with deterministic effects (line 3).

In some situations, planning in the HL may provide multiple plans, e.g., when the object that is to be grasped can be in one of multiple locations, tentative plans may be generated for the different hypotheses regarding the object's location. In such situations, all the HL plans are communicated to the

[3]We leave the proof of the correctness of this algorithm as future work.

Algorithm 1: Control loop of architecture

Input: The HL and LL domain representations, and the specific task for robot to perform.

1 LL observations reported to the HL history; HL initial state (s^H_{init}) communicated to LL.
2 Assign goal state s^H_{goal} based on task.
3 Generate HL plan(s).
4 **if** *multiple HL plans exist* **then**
5 Send plans to the LL, select plan with lowest (expected) action cost and communicate to the HL.
6 **end**
7 **if** *HL plan exists* **then**
8 **for** $a^H_i \in$ *HL plan:* $i \in [1,n]$ **do**
9 Pass a^H_i and relevant fluents to the LL.
10 Determine initial belief state over the relevant LL state space.
11 Generate the LL action policy.
12 **while** a^H_i *not completed* ***and*** a^H_i *achievable* **do**
13 Execute an action based on the LL action policy.
14 Make an LL observation and update belief state.
15 **end**
16 LL observations and action outcomes add statements to the HL history.
17 **if** *results unexpected* **then**
18 Perform diagnostics in the HL.
19 **end**
20 **if** *HL plan invalid* **then**
21 Replan in the HL (line 3).
22 **end**
23 **end**
24 **end**

LL and compared based on their costs, e.g., the expected time to execute the plans. The plan with the least expected cost is communicated to the HL (lines 4-6).

If an HL plan exists, actions are communicated one at a time to the LL along with the relevant fluents (line 9). For HL action a^H_i, the communicated fluents are used to automatically identify the relevant LL variables and set the initial belief state, e.g., a uniform distribution (line 10). An LL action policy is computed (line 11) and used to execute actions and update the belief state until a^H_i is achieved or inferred to be unachievable (lines 12-15). The outcome of executing the LL policy, and the LL observations, add to the HL history (line 16). For instance, if defined fluent *failure* is true for object ob_1 and room rm_1, the robot reports: $obs(loc(ob_1, rm_1), false)$ to the HL history. If the results are unexpected, diagnosis is performed in the HL (lines 17-19); we assume that the robot is capable of identifying these unexpected outcomes. If the HL plan is invalid, a new plan is generated (lines 20-22); else, the next action in the HL plan is executed.

4 Experimental setup and results

This section describes the experimental setup and results of evaluating the proposed architecture in indoor domains.

4.1 Experimental setup

The architecture was evaluated in simulation and on physical robots. To provide realistic observations in the simulator, we included object models that characterize objects using probabilistic functions of features extracted from images captured by a camera on physical robots (Li and Sridharan 2013). The simulator also uses action models that reflect the motion of the robot. Specific instances of objects of different classes were simulated in a set of rooms. The experimental setup also included an initial training phase in which the robot repeatedly executed the different movement actions and applied the visual input processing algorithms on images with known objects. A human participant provided some of the ground truth data, e.g., labels of objects in images. A comparison of the expected and actual outcomes was used to define the functions that describe the probabilistic transition diagram (T, O) in the LL, while the reward specification is defined by also considering the computational time required by different visual processing and navigation algorithms.

In each trial of the experimental results summarized below, the robot's goal is to move specific objects to specific places; the robot's location, target object, and locations of objects are chosen randomly in each trial. A sequence of actions extracted from an answer set obtained by solving the SPARC program of the HL domain representation provides an HL plan. If a robot (`robot1`) that is in the *office* is asked to fetch a textbook (`tb1`) from the *main_library*, the HL plan has the following sequence of actions:

$$move(robot1, main\_library)$$
$$grasp(robot1, tb1)$$
$$move(robot1, office)$$
$$putdown(robot1, tb1)$$

The LL action policies for each HL action are generated by solving the appropriate POMDP models using the APPL solver (Ong et al. 2010; Somani et al. 2013). In the LL, the location of an object is considered to be known with certainty if the belief (of the object's occurrence) in a grid cell exceeds a threshold (0.85).

We experimentally compared our architecture, with the control loop described in Algorithm 1, henceforth referred to as "PA", with two alternatives: (1) POMDP-1 (see Section 3.2); and (2) POMDP-2, which revises POMDP-1 by assigning high probability values to defaults to bias the initial belief states. These comparisons evaluate two hypotheses: (H1) PA enables a robot to achieve the assigned goals more reliably and efficiently than using POMDP-1; (H2) our representation of defaults improves reliability and efficiency in comparison with not using default knowledge or assigning high probability values to defaults.

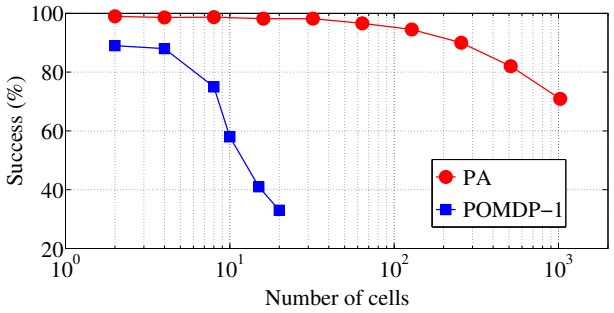


Figure 2: Ability to successfully achieve the assigned goal, as a function of the number of cells in the domain; with a limit on the time to compute policies PA significantly increases accuracy in comparison with just POMDP-1 as the number of cells in the domain increases.

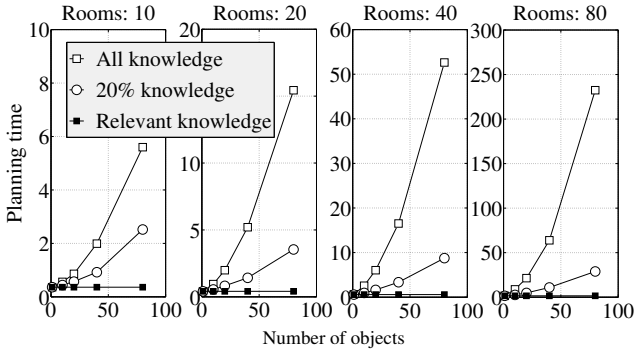


Figure 3: Planning time as a function of the number of rooms and the number of objects in the domain—*PA* scales to larger number of rooms and objects.

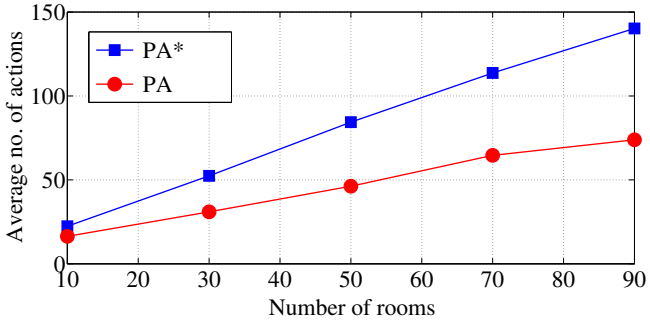


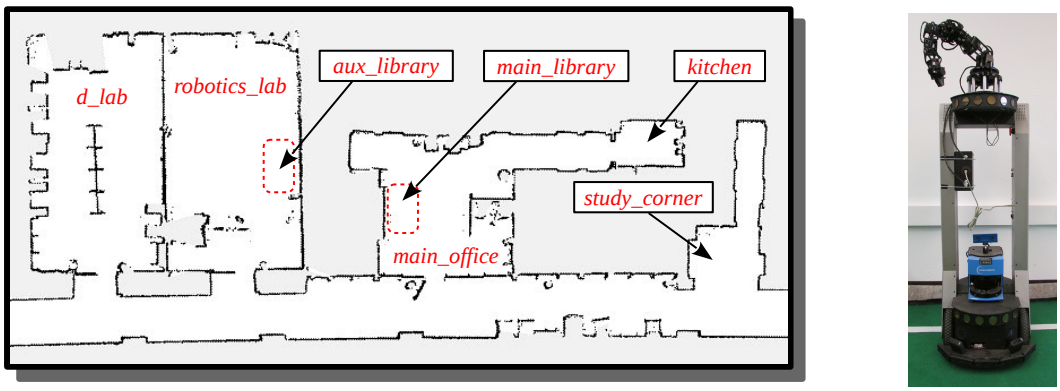
Figure 4: Effect of using default knowledge—principled representation of defaults significantly reduces the number of actions (and thus time) for achieving assigned goal.

4.2 Experimental Results

To evaluate H1, we first compared PA with POMDP-1 in a set of trials in which the robot's initial position is known but the position of the object to be moved is unknown. The solver used in POMDP-1 is given a fixed amount of time to compute action policies. Figure 2 summarizes the ability to successfully achieve the assigned goal, as a function of the number of cells in the domain. Each point in Figure 2 is the average of 1000 trials, and we set (for ease of interpretation) each room to have four cells. PA significantly improves the robot's ability to achieve the assigned goal in comparison with POMDP-1. As the number of cells (i.e., size of the domain) increases, it becomes computationally difficult to generate good POMDP action policies which, in conjunction with incorrect observations (e.g., false positive sightings of objects) significantly impacts the ability to successfully complete the trials. PA, on the other hand, focuses the robot's attention on relevant regions of the domain (e.g., specific rooms and cells). As the size of the domain increases, a large number of plans of similar cost may still be generated which, in conjunction with incorrect observations, may affect the robot's ability to successfully complete the trials—the impact is, however, much less pronounced.

Next, we computed the time taken by PA to generate a plan as the size of the domain increases. Domain size is characterized based on the number of rooms and the number of objects in the domain. We conducted three sets of experiments in which the robot reasons with: (1) all available knowledge of domain objects and rooms; (2) only knowledge relevant to the assigned goal—e.g., if the robot knows an object's default location, it need not reason about other objects and rooms in the domain to locate this object; and (3) relevant knowledge and knowledge of an additional 20% of randomly selected domain objects and rooms. Figure 3 summarizes these results. We observe that PA supports the generation of appropriate plans for domains with a large number of rooms and objects. We also observe that using only the knowledge relevant to the goal significantly reduces the planning time—such knowledge can be automatically selected using the relationships included in the HL system description. Furthermore, if we only use a probabilistic approach (POMDP-1), it soon becomes computationally intractable to generate a plan for domains with many objects and rooms; these results are not shown in Figure 3—see (Sridharan, Wyatt, and Dearden 2010; Zhang, Sridharan, and Washington 2013).

To evaluate H2, we first conducted multiple trials in which PA was compared with PA^*, a version that does not include any default knowledge. Figure 4 summarizes the average number of actions executed per trial as a function of the number of rooms in the domain—each sample point is the average of 10000 trials. The goal in each trial is (as before) to move a specific object to a specific place. We observe that the principled use of default knowledge significantly reduces the number of actions (and thus time) required to achieve the assigned goal. Next PA was compared with POMDP-2, which assigns high probability values to default information and suitably revises the initial belief state. We observe that the effect of assigning a probability value to defaults is arbitrary depending on multiple factors: (a) the numerical value chosen; and (b) whether the ground truth matches the default



(a) Domain map (b) Robot platform

Figure 5: Subset of the map of the second floor of our department; specific places are labeled as shown, and used during planning to achieve the assigned goals. The robot platform used in the experimental trials is also shown.

information. For instance, *if a large probability is assigned to the default knowledge that books are typically in the library, but the book the robot has to move is an exception to the default (e.g., a cookbook), it takes a significantly large amount of time for POMDP-2 to revise (and recover from) the initial belief*. PA, on the other hand, enables the robot to revise initial defaults and encode exceptions to defaults.

Robot Experiments: In addition to the trials in simulated domains, we compared PA with POMDP-1 on a wheeled robot over 50 trials conducted on two floors of our department building. This domain includes places in addition to those included in our illustrative example, e.g., Figure 5(a) shows a subset of the domain map of the third floor of our department, and Figure 5(b) shows the wheeled robot platform. Such domain maps are learned by the robot using laser range finder data, and revised incrementally over time. Manipulation by physical robots is not a focus of this work. Therefore, once the robot is next to the desired object, it currently asks for the object to be placed in the extended gripper; future work will include existing probabilistic algorithms for manipulation in the LL.

For experimental trials on the third floor, we considered 15 rooms, which includes faculty offices, research labs, common areas and a corridor. To make it feasible to use POMDP-1 in such large domains, we used our prior work on a hierarchical decomposition of POMDPs for visual sensing and information processing that supports automatic belief propagation across the levels of the hierarchy and model generation in each level of the hierarchy (Sridharan, Wyatt, and Dearden 2010; Zhang, Sridharan, and Washington 2013). The experiments included paired trials, e.g., over 15 trials (each), POMDP-1 takes 1.64 as much time as PA (on average) to move specific objects to specific places. For these paired trials, this 39% reduction in execution time provided by PA is statistically significant: $p\text{-}value = 0.0023$ at the 95% significance level.

Consider a trial in which the robot's objective is to bring a specific textbook to the place named *study_corner*. The robot uses default knowledge to create an HL plan that causes the robot to move to and search for the textbook in the *main_library*. When the robot does not find this textbook in the *main_library* after searching using a suitable LL policy, replanning in the HL causes the robot to investigate the *aux_library*. The robot finds the desired textbook in the *aux_library* and moves it to the target location. A video of such an experimental trial can be viewed online:
`http://youtu.be/8zL4R8te6wg`

5 Conclusions

This paper described a knowledge representation and reasoning architecture for robots that integrates the complementary strengths of declarative programming and probabilistic graphical models. The system descriptions of the tightly coupled high-level (HL) and low-level (LL) domain representations are provided using an action language, and the HL definition of recorded history is expanded to allow prioritized defaults. Tentative plans created in the HL using defaults and commonsense reasoning are implemented in the LL using probabilistic algorithms, generating observations that add suitable statements to the HL history. In the context of robots moving objects to specific places in indoor domains, experimental results indicate that the architecture supports knowledge representation, non-monotonic logical inference and probabilistic planning with qualitative and quantitative descriptions of knowledge and uncertainty, and scales well as the domain becomes more complex. Future work will further explore the relationship between the HL and LL transition diagrams, and investigate a tighter coupling of declarative logic programming and probabilistic reasoning for robots.

Acknowledgments

The authors thank Evgenii Balai for making modifications to SPARC to support some of the experiments reported in this paper. This research was supported in part by the U.S. Office of Naval Research (ONR) Science of Autonomy Award

N00014-13-1-0766. Opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the ONR.

References

Balai, E.; Gelfond, M.; and Zhang, Y. 2013. Towards Answer Set Programming with Sorts. In *International Conference on Logic Programming and Nonmonotonic Reasoning*.

Balduccini, M., and Gelfond, M. 2003. Logic Programs with Consistency-Restoring Rules. In *Logical Formalization of Commonsense Reasoning, AAAI Spring Symposium Series*, 9–18.

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming* 9(1):57–144.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Chen, X.; Xie, J.; Ji, J.; and Sui, Z. 2012. Toward Open Knowledge Enabling for Human-Robot Interaction. *Journal of Human-Robot Interaction* 1(2):100–117.

Erdem, E.; Aker, E.; and Patoglu, V. 2012. Answer Set Programming for Collaborative Housekeeping Robotics: Representation, Reasoning, and Execution. *Intelligent Service Robotics* 5(4).

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning and the Design of Intelligent Agents*. Cambridge University Press.

Gelfond, M. 2008. Answer Sets. In Frank van Harmelen and Vladimir Lifschitz and Bruce Porter., ed., *Handbook of Knowledge Representation*. Elsevier Science. 285–316.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. San Francisco, USA: Morgan Kaufmann.

Halpern, J. 2003. *Reasoning about Uncertainty*. MIT Press.

Hanheide, M.; Gretton, C.; Dearden, R.; Hawes, N.; Wyatt, J.; Pronobis, A.; Aydemir, A.; Gobelbecker, M.; and Zender, H. 2011. Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour. In *International Joint Conference on Artificial Intelligence*.

Hoey, J.; Poupart, P.; Bertoldi, A.; Craig, T.; Boutilier, C.; and Mihailidis, A. 2010. Automated Handwashing Assistance for Persons with Dementia using Video and a Partially Observable Markov Decision Process. *Computer Vision and Image Understanding* 114(5):503–519.

Kaelbling, L., and Lozano-Perez, T. 2013. Integrated Task and Motion Planning in Belief Space. *International Journal of Robotics Research* 32(9-10).

Laird, J. E.; Newell, A.; and Rosenbloom, P. 1987. SOAR: An Architecture for General Intelligence. *Artificial Intelligence* 33(3).

Langley, P., and Choi, D. 2006. An Unified Cognitive Architecture for Physical Agents. In *The Twenty-first National Conference on Artificial Intelligence (AAAI)*.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic* 7(3):499–562.

Li, X., and Sridharan, M. 2013. Move and the Robot will Learn: Vision-based Autonomous Learning of Object Models. In *International Conference on Advanced Robotics*.

Milch, B.; Marthi, B.; Russell, S.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2006. BLOG: Probabilistic Models with Unknown Objects. In *Statistical Relational Learning*. MIT Press.

Ong, S. C.; Png, S. W.; Hsu, D.; and Lee, W. S. 2010. Planning under Uncertainty for Robotic Tasks with Mixed Observability. *International Journal of Robotics Research* 29(8):1053–1068.

Richardson, M., and Domingos, P. 2006. Markov Logic Networks. *Machine learning* 62(1).

Rosenthal, S., and Veloso, M. 2012. Mobile Robot Planning to Seek Help with Spatially Situated Tasks. In *National Conference on Artificial Intelligence*.

Sanner, S., and Kersting, K. 2010. Symbolic Dynamic Programming for First-order POMDPs. In *National Conference on Artificial Intelligence (AAAI)*.

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. S. 2013. DESPOT: Online POMDP Planning with Regularization. In *Advances in Neural Information Processing Systems (NIPS)*.

Sridharan, M.; Wyatt, J.; and Dearden, R. 2010. Planning to See: A Hierarchical Aprroach to Planning Visual Actions on a Robot using POMDPs. *Artificial Intelligence* 174:704–725.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for Human-Robot Teaming in Open Worlds. *ACM Transactions on Intelligent Systems and Technology* 1(2):14:1–14:24.

Zhang, S.; Sridharan, M.; and Bao, F. S. 2012. ASP+POMDP: Integrating Non-monotonic Logical Reasoning and Probabilistic Planning on Robots. In *International Joint Conference on Development and Learning and on Epigenetic Robotics*.

Zhang, S.; Sridharan, M.; and Washington, C. 2013. Active Visual Planning for Mobile Robot Teams using Hierarchical POMDPs. *IEEE Transactions on Robotics* 29(4).

Planning a Robot's Search for Multiple Residents in a Retirement Home Environment

Markus Schwenk[1,2] and **Tiago Vaquero**[1] and **Goldie Nejat**[1] and **Kai O. Arras**[2]
[1]Autonomous Systems and Biomechatronics Laboratory
Department of Mechanical and Industrial Engineering, University of Toronto, Canada
[2]Social Robotics Laboratory
Department of Computer Science, University of Freiburg, Germany
markus.schwenk@mail.utoronto.ca, {tvaquero, nejat}@mie.utoronto.ca, arras@informatik.uni-freiburg.de

Abstract

In this paper we address the planning problem of a robot searching for multiple residents in a retirement home in order to remind them of an upcoming multi-person recreational activity before a given deadline. We introduce a novel Multi-User Schedule Based (M-USB) Search approach which generates a high-level plan to maximize the number of residents that are found within the given time frame. From the schedules of the residents, the layout of the retirement home environment as well as direct observations by the robot, we obtain spatio-temporal likelihood functions for the individual residents. The main contribution of our work is the development of a novel approach to compute a reward to find a search plan for the robot using: 1) the likelihood functions, 2) the availabilities of the residents, and 3) the order in which the residents should be found. Simulations were conducted on a floor of a real retirement home to compare our proposed M-USB Search approach to a Weighted Informed Walk and a Random Walk. Our results show that the proposed M-USB Search finds residents in a shorter amount of time by visiting fewer rooms when compared to the other approaches.

1 Introduction

The health and quality of life of older adults living in long-term care facilities can be improved by these individuals engaging in stimulating recreational activities such as playing games, playing musical instruments, doing crossword puzzles, or reading (Menec 2003). These types of activities can delay age-related health decline (Bath and Deeg 2005) and prevent social isolation (Findlay 2003), which could potentially decrease the risk of dementia in elder adults (Wilson et al. 2007). However, the lack of these activities in eldercare facilities (PriceWaterCoopers LLP 2001) exists due to a shortage of healthcare workers (Sharkey 2008), which could be aggravated in the near future due to the rapid growth of the elderly population (Centre for Health Workforce Studies 2006). Socially assistive robots have been shown to be a promising technology to assist the elderly and to support caregivers in eldercare facilities (Oida et al. 2011; McColl, Louie, and Nejat 2013).

This is an extended version of our paper in AAAI 2014

Our research focuses on the development of socially assistive robots that can autonomously organize and facilitate group-based recreational activities for the elderly. In this paper, we address the planning problem of a robot searching for multiple residents in a retirement home environment in order to invite and remind them of an upcoming multi-person recreational activity. The robot's objective is to maximize the number of residents it finds in a given time frame before the activity starts. During the search, the robot has to consider that the residents have their own schedules which contain appointments in different rooms of the environment, during which the residents are not always available for interaction with the robot. Based on these schedules, the robot also considers the order in which the residents have to be found to avoid searching for unavailable people. We introduce a novel Multi-User Schedule Based (M-USB) Search method which plans the robot's search for a set of non-static residents in a structured environment within a given time frame based on the residents' daily schedules.

Robotic search for people in structured environments has been investigated in the literature for different scenarios. For example, in (Elinas, Hoey, and Little 2003), the robot HOMER was designed to deliver messages one at a time to a particular person in a workspace environment. The robot stored a likelihood function for each person's location and performed a best-first search. The search consisted of visiting the nearest location to the robot based on the likelihood function for a particular person. If the person was not found, the robot iteratively visited other rooms until either the person was found or all regions had been visited. For these scenarios, a person was assumed to be at a static location in the environment. The search for multiple static targets in an indoor environment has been addressed in (Lau, Huang, and Dissanayake 2005). A dynamic programming approach was used to plan the search on a topological ordered graph, using a probability distribution which models the probability of meeting one of the targets in a given room at a given time. In (Tipaldi and Arras 2011), a robot's ability to blend itself into the workflows of people within human office environments was addressed. The authors developed a spatial Poisson process which was learned from observations of people to represent spatio-temporal patterns of human activities. A Markov Decision Process (MDP) Model was used to generate a robot's path through the environment to maximize the

probability of encountering a person. In (Lau, Huang, and Dissanayake 2006), the problem of a robot searching for a moving target within an indoor environment has been addressed. The Optimal Searcher Path (OSP) Problem, which models the search for static targets as sequentially searching in adjacent equal-sized parts of an environment, was extended to: 1) handle different room sizes, and 2) to search for a single moving target whose movements were modelled as a Markov Process. A probability distribution over the different regions was used to express the knowledge about the target's location. A branch and bound method was proposed in order to plan a sequence of regions the robot had to visit in order to maximize the robot's chances of finding the target in a given amount of time.

Uniquely our work addresses the robotic search problem of finding a specific set of multiple moving residents in a retirement home setting considering their individual daily schedules. Such schedule-based multi-user search for non-static people has not yet been addressed in the literature. To address this problem, we obtain spatio-temporal likelihood functions for every resident of the retirement home. We propose an approach to generate these resident likelihood functions as a composition of: 1) the residents' schedules, 2) direct observations by the robot in the environment, and 3) the layout of the environment. To do this, a weighting is applied to the above sources of information based on their time-dependent certainties of predicting a person's location. The main contribution of our work is the development of an MDP planner that uses a novel approach to compute the reward to determine the robot's search plan for finding multiple residents using: 1) the resident likelihood functions, 2) the availabilities of the individual residents, and 3) the order in which the residents should be found.

The paper is organized as follows. In section 2 we introduce and formalize the M-USB Search approach. We describe the resident likelihood functions and define an MDP Model and an algorithm which generates the robot's search plan. In section 3 we describe the simulated experiments as well as the experiment results and discussion. Concluding remarks are presented in section 4.

Environment. We model the environment as a set of regions $\mathbb{R} \in \mathbb{R}^E$ (e.g. rooms, corridors, common areas) in which the search takes place. For each person $p \in \mathbb{P}$, where $\mathbb{P}$ is the set of all residents who are living in the environment, we assign one region of the environment as solely that person's: his/her private room. For each region, a room-class $\mathbb{C} = class(\mathbb{R}) \in \mathbb{C}^E$ is assigned, e.g. "Common Room", "Bedroom", "Corridor", or "Dining Hall". The room-classes depend on the activities the residents engage in when they are in these regions. The room-class $\mathbb{C}_{private} \in \mathbb{C}^E$ in particular is the room-class which contains all private rooms. We define $regions(\mathbb{C})$ to contain all regions $\mathbb{R}$ with $\mathbb{C} = class(\mathbb{R})$.

2 M-USB Search

The Multi-User Schedule Based Search presented in this work is a planning procedure that provides a plan $\mathcal{P}^*$ for a robot to find as many people as possible from a given set of target people in a given order within a defined time frame. The retirement home environment is defined to consist of several different regions which represent the topology of the building (e.g., rooms and corridors). The plan $\mathcal{P}^*$ will include a sequence of actions which model the whole search process. The possible actions are: *drive* which lets the robot travel from one region to another; *rest* which lets the robot rest for a short period of time; and *search* in which the robot executes a low-level search procedure in a specific region (e.g., frontier exploration, random walk). We use backwards induction to compute $\mathcal{P}^*$ based on a Markov Decision Process which models the search using the aforementioned actions. To obtain a reward for this MDP, we setup a likelihood function for each person which models the probability that the person is in a specific region at a given time of the day. The likelihood functions of the individual residents are combined to generate a reward which respects the target residents' availability constraints (obtained from their schedules) and the order in which the residents should be found.

This paper will present the computation of the plan $\mathcal{P}^*$. This plan is to be executed by a mobile socially assistive robot that can navigate the environment as well as detect and recognize individual residents. Once a specific person is found, the robot greets the person and verbally provides a reminder to him/her. We assume a local-search routine already exists on the robot which allows the robot to search for people in a given region. Once a person has been detected,

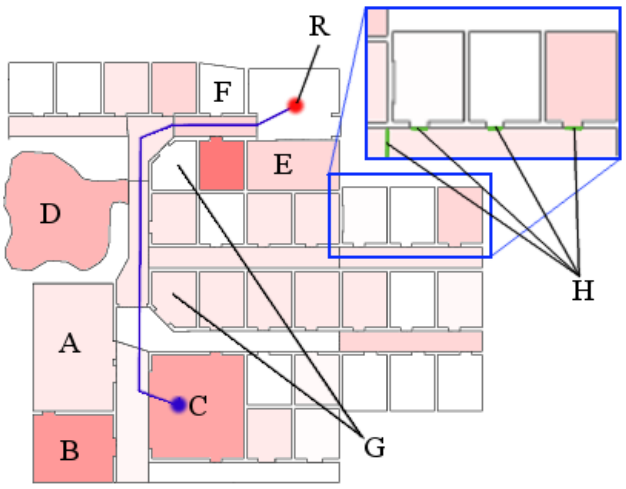


Figure 1: The map of the simulated retirement home with the rooms: dining hall (A), games room (B), TV-room (C), garden (D), nurse station (E), family visit room (F) and shower rooms (G). All other rooms are private rooms and corridors. The color of the regions indicates the current reward for each region (red: high, white: low) of an example scenario. The current generated plan has the robot (R) driving to the TV-room (blue trajectory) where it starts a local search. (H) shows the crossable edges (doors) in the scaled portion of the environment.

the robot has to compute a new plan (replanning).

Problem Setup

Each region can be represented as a polygon. The edges of this polygon can be marked as "crossable" if there is no physical border at an edge (e.g., if the edge represents a doorway). We define $neighbours(\mathbb{R})$ to be all regions which share a crossable edge with $\mathbb{R}$, i.e., a person or a robot can move from $\mathbb{R}$ to any region $\mathbb{R}_n \in neighbours(\mathbb{R})$ without entering a third region. Figure 1 shows an example environment.

Schedules and Availability of Residents. For each person $p \in \mathbb{P}$, we consider a schedule which defines all of his/her appointments on a given day. An appointment has a start time and an end time, and is assigned to a region $\mathbb{R}$ in which it takes place. We model the availability of each person $p \in \mathbb{P}$ as function $\beta_p(t)$ such that $\beta_p(t) = 1$ if p is available at time t and $\beta_p(t) = 0$ otherwise.

Motion Model. We assume a simple motion model for the residents. We model the probability $p_m(\mathbb{R}_1, \mathbb{R}_2, p, \Delta t)$ that person p has moved from region $\mathbb{R}_1$ to region $\mathbb{R}_2$ in the time frame Δt. This probability can be obtained from the person's speed v_p and the distance $d(\mathbb{R}_1, \mathbb{R}_2)$ between the two regions. We define the set $\mathbb{R}^r(\mathbb{R}, p, \Delta t)$ which contains all regions which the person could have entered:

$$\mathbb{R}^r(\mathbb{R}, p, \Delta t) = \{\mathbb{R}' \,|\, d(\mathbb{R}, \mathbb{R}') \leq v_p \cdot \Delta t\}\,. \tag{1}$$

The value of $p_m(\mathbb{R}_1, \mathbb{R}_2, p, \Delta t)$ can then be obtained as:

$$p_m(\mathbb{R}_1, \mathbb{R}_2, p, \Delta t) = \begin{cases} \mu, & \text{if } \mathbb{R}_2 \in \mathbb{R}^r(\mathbb{R}_1, p, \Delta t) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

with $\mu = |\mathbb{R}^r(\mathbb{R}_1, p, \Delta t)|^{-1}$. This motion model considers all possible movements of the resident within the environment.

Search Query. When the robot receives a query q, it is to find a set of residents $p \in \mathbb{P}_q \subseteq \mathbb{P}$ within a given deadline t_{max}. Query q also specifies the order in which the residents should be found.

Setting up Resident Likelihood Functions

For each resident $p \in \mathbb{P}$, we can set up a likelihood function $\mathcal{L}(p, \mathbb{R}, t)$ which represents the probability that p is in region $\mathbb{R}$ at time t. This individual likelihood function is composed of four different weighted likelihood functions $\mathcal{L}_k(p, \mathbb{R}, t)$, with $k \in \{s, lkrl, l, env\}$. These likelihood functions are: $\mathcal{L}_s$ which is obtained by analyzing the resident's schedule; $\mathcal{L}_{lkrl}$ which is based on the last known location of the resident; $\mathcal{L}_l$ which describes the resident's behaviour that the robot has learned; and $\mathcal{L}_{env}$ which can be obtained from the structure of the environment. We will discuss these likelihood functions in the following sections. As convention, we define $0 \leq \mathcal{L}_k \leq 1$ and $\sum_{\mathbb{R}} \mathcal{L}_k(p, \mathbb{R}, t) = 1$ for each person $p \in \mathbb{P}$ and each likelihood function $\mathcal{L}_k$. A value $\mathcal{L}_k(p, \mathbb{R}, t) = 1$ means that the person is in $\mathbb{R}$ at time t while $\mathcal{L}_k(p, \mathbb{R}, t) = 0$ indicates that the person cannot be in the region at this time.

Schedule Analyzer. We model $\mathcal{L}_s(p, \mathbb{R}, t)$ using the schedule of resident p. Assuming that with a probability of $0 \leq p_a \leq 1$ the person participates in an appointment defined in his/her schedule, we set $\mathcal{L}_s(p, \mathbb{R}, t) = p_a$ for the time frame of the appointment for the region assigned to this particular appointment, and $\mathcal{L}_s(p, \mathbb{R}, t) = \frac{(1-p_a)}{|\mathbb{R}^E|-1}$ for all other regions. For any time t_k between two appointments where the last appointment ends at time t_{k-1} and the next appointment starts at t_{k+1}, we define $\alpha_{k-1} = \frac{t_{k+1}-t_k}{t_{k+1}-t_{k-1}}$ and $\alpha_{k+1} = \frac{t_k-t_{k-1}}{t_{k+1}-t_{k-1}}$. If there is no next appointment, we set $\alpha_{k+1} = 0$ and $\alpha_{k-1} = 1$. If there is no previous appointment, $\alpha_{k+1} = 1$ and $\alpha_{k-1} = 0$. We can then define the likelihood function to be:

$$\begin{aligned} &\mathcal{L}_s(p, \mathbb{R}, t_k) = \\ &\quad \alpha_{k-1} \cdot \sum_{\mathbb{R}' \in \mathbb{R}^{\mathbb{E}}} \mathcal{L}_s(p, \mathbb{R}', t_{k-1}) \cdot p_m(\mathbb{R}', \mathbb{R}, p, t_k - t_{k-1}) + \\ &\quad \alpha_{k+1} \cdot \sum_{\mathbb{R}' \in \mathbb{R}^{\mathbb{E}}} \mathcal{L}_s(p, \mathbb{R}', t_{k+1}) \cdot p_m(\mathbb{R}, \mathbb{R}', p, t_{k+1} - t_k) \end{aligned} \tag{3}$$

Last Known Resident Location. To be able to take into account when the robot last detected a person earlier that day who it is currently searching for, we set up a database which stores the time t_d^p and region $\mathbb{R}_d^p$ of the last detection of person p. Using the motion model of the resident, this information can be used to generate $\mathcal{L}_{lkrl}(p, \mathbb{R}, t)$:

$$\mathcal{L}_{lkrl}(p, \mathbb{R}, t) = p_m(\mathbb{R}_d^p, \mathbb{R}, p, t - t_d^p). \tag{4}$$

If the person has not been previously detected, we assign a uniform distribution $\mathcal{L}_{lkrl}(p, \mathbb{R}, t) = \frac{1}{|\mathbb{R}^E|}$.

Learned Behaviour. A person's behaviour, which is not defined in the schedule (e.g., a person often takes a walk in the garden after lunch) but which has been learned by the robot based on its observations is also stored by the robot as $\mathcal{L}_l(p, \mathbb{R}, t_k)$, where t_k indicates a time step. If no data from the previous days exist, we assign a uniform distribution $\mathcal{L}_l(p, \mathbb{R}, t_k) = \frac{1}{|\mathbb{R}^E|}$. During the day the robot remembers whether it detected person p in time step t_k in region $\mathbb{R}$. It saves this information in $g(p, \mathbb{R}, t_k)$ which is either 1 when the person has been detected or 0 otherwise. We define:

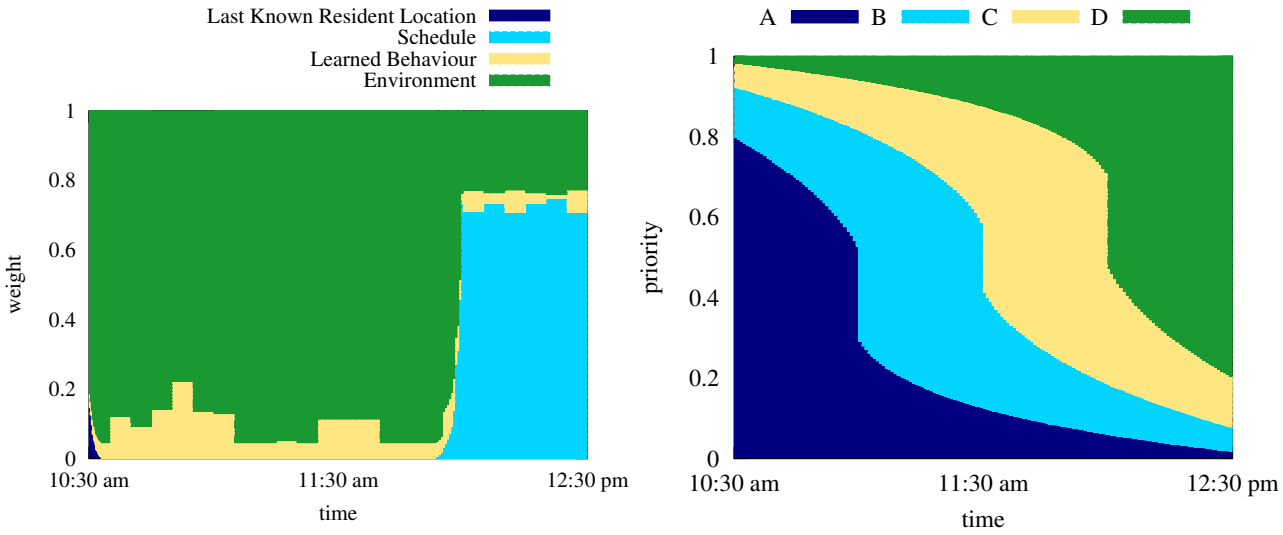
$$f(p, t_k) = \sum_{\mathbb{R} \in \mathbb{R}^E} g(p, \mathbb{R}, t_k) \tag{5}$$

to be the number of regions in which person p has been detected in time step t_k. At the end of the day the updated likelihood function evolves to:

$$\mathcal{L}'_l(p, \mathbb{R}, t_k) = \alpha \cdot \frac{g(p, \mathbb{R}, t_k)}{f(p, t_k)} + (1 - \alpha) \cdot \mathcal{L}_l(p, \mathbb{R}, t_k) \tag{6}$$

with $0 \leq \alpha \leq 1$ if $f(p, t_k) \geq 1$ and $\mathcal{L}'_l(p, \mathbb{R}, t_k) = \mathcal{L}_l(p, \mathbb{R}, t_k)$ otherwise.

Environment. The topology of the environment can be used to generate $\mathcal{L}_{env}(p, \mathbb{R}, t)$. We obtain $\mathbb{C}^p_{private}$ by removing the private room of p from $\mathbb{C}_{private}$ and define a



(a) Weights for one Person

(b) Priority Function.

Figure 2: (a) Weights for one person during the time frame of 10:30 am to 12:30 pm. At 12:00 pm the person has a one-hour appointment which gives the schedule a higher weight at this time. (b) Priority function for four residents who are all available in the considered time frame.

new room-class $\mathbb{C}^{p'}_{private}$ for the resident p, containing only his/her private room. We define $\mathbb{C}^p$ to be:

$$\mathbb{C}^p = \mathbb{C}^E \setminus \{\mathbb{C}_{private}\} \bigcup \left\{\mathbb{C}^p_{private}, \mathbb{C}^{p'}_{private}\right\}. \quad (7)$$

For each room-class $\mathbb{C} \in \mathbb{C}^p$, we define a probability $p_{\mathbb{C}}(p)$ to be the probability that person p is in one of the rooms in $regions(\mathbb{C})$. Assuming that a person will spend most of her/his spare time either in her/his private room or in the common rooms, we assign higher values for $\mathbb{C}^{p'}_{private}$ and the room-class containing the common rooms. We apply the constraint:

$$\sum_{\mathbb{C} \in \mathbb{C}^p} p_{\mathbb{C}}(p) = 1. \quad (8)$$

We define $\mathcal{L}_{env}(p, \mathbb{R}, t)$ to be:

$$\mathcal{L}_{env}(p, \mathbb{R}, t) = \frac{p_{class(\mathbb{R})}(p)}{|regions(class(\mathbb{R}))|} \quad (9)$$

with $class(\mathbb{R}) \in \mathbb{C}^p$. $\mathcal{L}_{env}(p, \mathbb{R}, t)$ is constant over the day.

Pre-computation. Since the schedules, the topology of the environment, and the learned behaviour remain the same once they are provided to the robot at the beginning of a day, $\mathcal{L}_s(p, \mathbb{R}, t)$, $\mathcal{L}_l(p, \mathbb{R}, t)$, and $\mathcal{L}_{env}(p, \mathbb{R}, t)$ can be computed before a search query is received. $\mathcal{L}_{lkrl}(p, \mathbb{R}, t)$ is computed dynamically when the query is received.

Combining the Likelihood Functions for one Person. The four likelihood functions $\mathcal{L}_k(p, \mathbb{R}, t)$ can be combined to generate $\mathcal{L}(p, \mathbb{R}, t)$. As the certainties with which the four likelihood functions can predict a resident's location differ (e.g., the Last Known Resident Location will have high uncertainty when the person has not been detected for several hours, and the Schedule Analyzer will have high certainty when the person has an appointment), a weighting function can be used. The certainty of one likelihood function $\mathcal{L}_k(p, \mathbb{R}, t)$ can be represented by its variance:

$$\mathrm{Var}(\mathcal{L}_k(p, \mathbb{R}, t)) = \frac{1}{|\mathbb{R}^E|} \cdot \sum_{\mathbb{R} \in \mathbb{R}^E} \left[\mathcal{L}_k(p, \mathbb{R}, t) - \frac{1}{|\mathbb{R}^E|}\right]^2. \quad (10)$$

For each likelihood function $\mathcal{L}_k$, we introduce the weight w_k at time t:

$$w_k(t) = \frac{\mathrm{Var}(\mathcal{L}_k(p, \mathbb{R}, t))}{\sum_k \mathrm{Var}(\mathcal{L}_k(p, \mathbb{R}, t))} \quad (11)$$

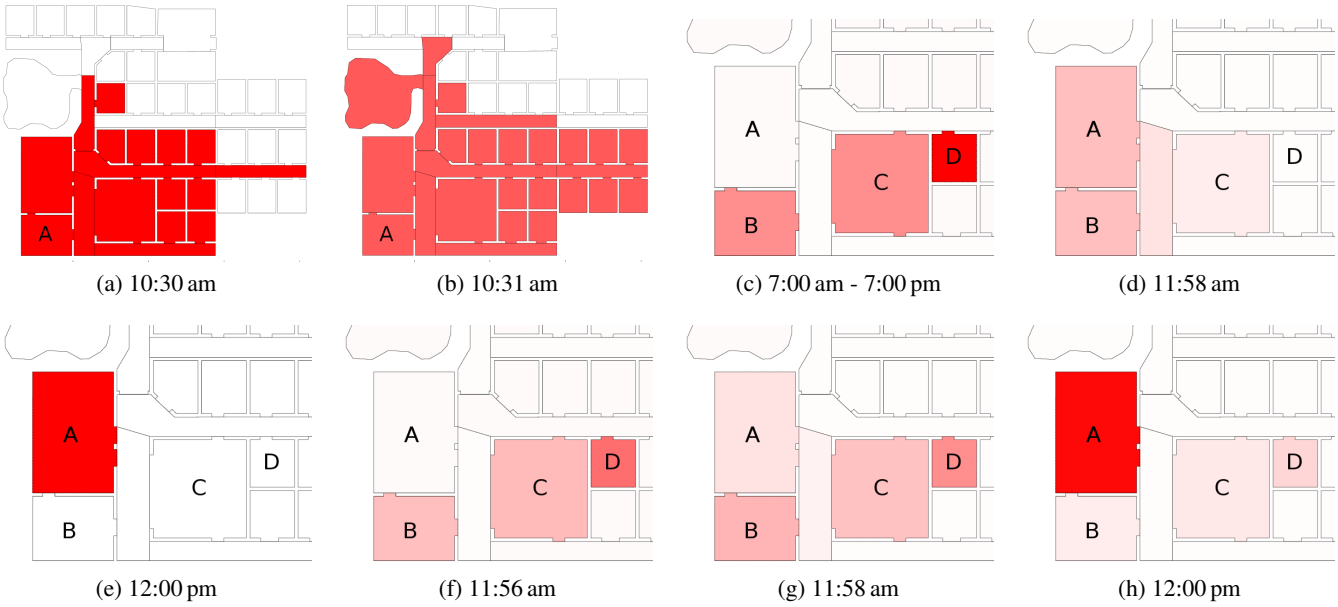
where $\sum_k w_k(t) = 1$. The final combined likelihood function is defined to be:

$$\mathcal{L}(p, \mathbb{R}, t) = \sum_k w_k(t) \cdot \mathcal{L}_k(p, \mathbb{R}, t). \quad (12)$$

Figure 2(a) shows an example of four weights used for one resident. Figure 3 shows examples for the different likelihood functions as well as the combined likelihood function for one resident.

Modelling the Transition System

The objective is to find a sequence of actions the robot should execute in order to find as many persons as possible in $\mathbb{P}_q$ within the given deadline t_{max}. We model the search as a Markov Decision Process. We discretize time using time steps of duration Δt which is the execution time for each individual action. The M-USB Search is modelled to consist of three possible action sequences that the robot can perform



(a) 10:30 am (b) 10:31 am (c) 7:00 am - 7:00 pm (d) 11:58 am

(e) 12:00 pm (f) 11:56 am (g) 11:58 am (h) 12:00 pm

Figure 3: Different likelihood functions for one resident (dark red: high likelihood, white: low likelihood). (a, b) $\mathcal{L}_{lkrl}(p, \mathbb{R}, t)$ three and four minutes after the person has been detected in the Games Room (A) at 10:27 am, (c) $\mathcal{L}_{env}(p, \mathbb{R}, t)$ for the resident living in D. B and C are common rooms. (d, e) $\mathcal{L}_s(p, \mathbb{R}, t)$ when the resident has an appointment in the Dining Hall (A) at 12 pm, and (f-h) the combined resident likelihood function $\mathcal{L}(p, \mathbb{R}, t)$ before and during the appointment.

in each region: 1) $rest$ in which the robot rests for one time step, 2) $search$ in which the robot performs a local search within the region, and 3) $drive$ in which the robot drives to one of the neighbouring regions. Since the time it takes to perform a local search within a region depends on the geometry of the region, we introduce $\mathbb{T}^s_k$ to represent the number of time steps Δt a search within region $\mathbb{R}_k$ takes. For region $\mathbb{R}_l \in neighbours(\mathbb{R}_k)$, we define $\mathbb{T}^d_{k,l}$ as being the number of time steps the transition between $\mathbb{R}_k$ and $\mathbb{R}_l$ takes.

The states $s \in \mathbb{S}$ of the MDP represent the states of the robot, which depend on the region the robot is in and the current action sequence it is performing. We define the following sets of states $\mathbb{S}^s_k$ and $\mathbb{S}^d_k$ to contain all states during the $search$ and $drive$ sequences for each region $\mathbb{R}_k \in \mathbb{R}^E$:

1. $\mathbb{S}^s_k = \{search^t_k\}$ with $0 \leq t < \mathbb{T}^s_k - 1$, and
2. $\mathbb{S}^d_k = \bigcup_{\mathbb{R}_l} \left\{drive^t_{k,l}\right\}$ with $0 \leq t < \mathbb{T}^d_{k,l} - 1$, $\mathbb{R}_l \in neighbours(\mathbb{R}_k)$.

In addition to the aforementioned states within the $search$ and $drive$ sequences, we define the state s'_k for each region $\mathbb{R}_k$ as the robot's state: 1) after a region has been entered by any drive sequence, 2) after the rest action has been executed, 3) after the full search sequence of $\mathbb{R}_k$ has been executed, or 4) when the robot is creating a new plan when being in $\mathbb{R}_k$. The set of all possible states for region $\mathbb{R}_k$ is defined to be:

$$\mathbb{S}_k = \{s'_k\} \cup \mathbb{S}^s_k \cup \mathbb{S}^d_k. \tag{13}$$

We define the following robot actions α for each region $\mathbb{R}_k \in \mathbb{R}^E$:

1. $\mathbb{A}^r_k = \{rest_k\}$,
2. $\mathbb{A}^s_k = \{search^t_k\}$ with $0 \leq t < \mathbb{T}^s_k$, and
3. $\mathbb{A}^d_k = \bigcup_{\mathbb{R}_l} \left\{drive^t_{k,l}\right\}$ with $0 \leq t < \mathbb{T}^d_{k,l}$, $\mathbb{R}_l \in neighbours(\mathbb{R}_k)$.

For each region $\mathbb{R}_k$, the set of all possible actions is:

$$\mathbb{A}_k = \mathbb{A}^r_k \cup \mathbb{A}^s_k \cup \mathbb{A}^d_k. \tag{14}$$

Transitions between the states describe how a robot state changes when it performs a particular action. In particular, the successor $succ(\alpha)$ of action α is defined as the state which follows α. The overall transition system is shown in Figure 4.

For each action a time-dependent reward $R(\alpha, t)$ is assigned. This reward is evaluated to compute the plan $\mathcal{P}^*$ and depends on the region in which the action is performed. Therefore, we define a region to each action: $\mathbb{R} = region(\alpha)$. For each action α in $\mathbb{A}^r_k$ and $\mathbb{A}^s_k$, we define $region(\alpha) = \mathbb{R}_k$. For each pair of neighbouring regions $\mathbb{R}_k$ and $\mathbb{R}_l$, we define $\mathbb{T}^{d,cross}_{k,l}$ to be the number of time steps after which the region $\mathbb{R}_l$ is entered during a drive sequence from $\mathbb{R}_k$ to $\mathbb{R}_l$. We then define $region(\alpha) = \mathbb{R}_k$ if $t < \mathbb{T}^{d,cross}_{k,l}$ and $region(\alpha) = \mathbb{R}_l$ if $t \geq \mathbb{T}^{d,cross}_{k,l}$ for each drive action α.

Resident Detection Probability of Actions. Assuming that during a search the probability that a person who is in the searched region is detected is different from the probability that a person is detected while the robot is

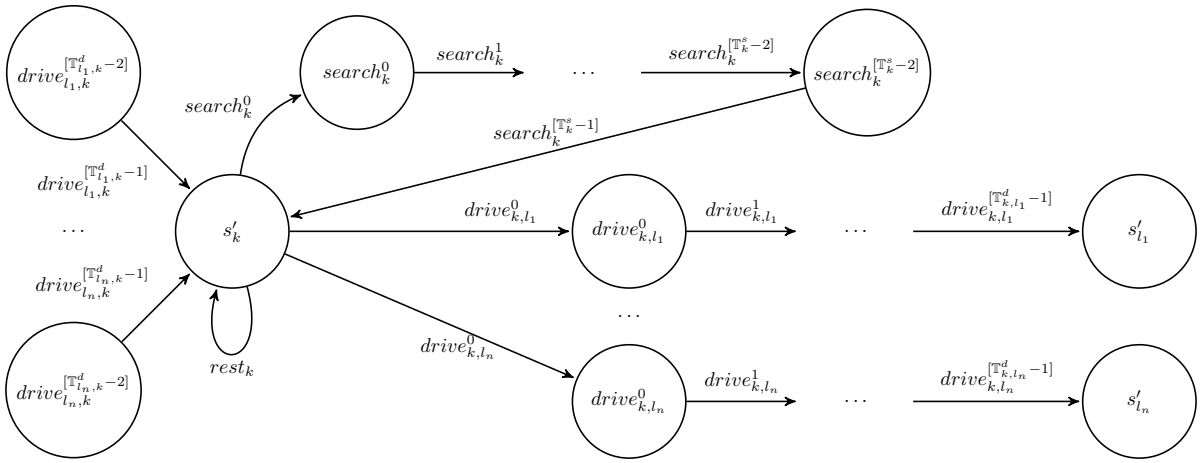


Figure 4: The transition system for an arbitrary region k with neighbours $l_1, \ldots, l_n$. Being in state s'_k, the robot can either *rest* one time step, start a full *search* sequence or a *drive* sequence to one of the neighbouring regions l_i with $i = 1, \ldots, n$ which leads to state s'_{l_i}. *Search* sequences and the *rest* action in region k lead to s'_k.

just driving between two regions or resting in one region, we define an attractivity $\delta(\alpha)$ with $0 \leq \delta(\alpha) \leq 1$ for each action α. This attractivity depends on the geometry of the region $\mathbb{R} = region(\alpha)$. We define the attractivity $\delta(\alpha_s) = \mathcal{A}(region(\alpha_s))^{-1}$ for each search action α_s with $\mathcal{A}(region(\alpha_s))$ being the area of $region(\alpha_s)$. We define $\delta(\alpha_r) = a \cdot \delta(\alpha_s)$ with $0 < a \leq 1$ for each rest action and $\delta(\alpha_d) = b \cdot \delta(\alpha_s)$ with $0 \leq b \leq 1$ for each drive action. If the robot detects residents with higher probability while driving then $b > a$ holds; $a > b$ holds otherwise.

Modelling the Order and Availability of Residents

The order in which the residents should be found is given and has been obtained from the persons' schedules to avoid searching for unavailable residents. The robot should try to keep this order if possible. However, if the robot can maximize the number of people found by changing the order, it can also do so. To search for the residents $p \in \mathbb{P}_q$ in the given order, we introduce a priority function $\pi_p(t)$ with $0 \leq \pi_p(t) \leq 1$ for each person $p \in \mathbb{P}_q$ and apply the following constraints:

$$\sum_{p \in \mathbb{P}_q} \pi_p(t) \cdot \beta_p(t) = 1 \; \forall t \tag{15}$$

and

$$\int_{t_0}^{t_{max}} \pi_p(t) \cdot \beta_p(t)\, dt = \frac{t_{max} - t_0}{|\mathbb{P}_q|} \;\; \forall p \tag{16}$$

which are used to ensure that the same search effort is applied to all residents during the search process. We model $\pi_p(t)$ to provide a high priority to the time interval assigned to the resident p based on the given order. However, to allow the robot to search for other residents $p' \in \mathbb{P}_q$ during this time interval, we allow $\pi_{p'}(t) \neq 0$ when $\beta_{p'}(t) = 1$. Figure 2(b) shows such a priority function for four people A, B, C, and D to be searched in this order.

Finding $\mathcal{P}^*$

In order to find the set of residents within the given deadline, we define a reward for each action of the MDP model of the search. The reward is based on the resident likelihood functions and the availabilities of the residents in $\mathbb{P}_q$, the aforementioned priority functions and the attractivities:

$$R(\alpha, t) = \delta(\alpha) \cdot \sum_{p \in \mathbb{P}_q} \pi_p(t) \cdot \beta_p(t) \cdot \mathcal{L}(p, region(\alpha), t). \tag{17}$$

Since a deadline t_{max} is given, the search evolves to be a finite horizon MDP which can be solved using backwards induction (Tipaldi and Arras 2011). In particular, the utility $U_t(s)$ is evaluated for each possible state s at time t using the Bellman equation:

$$U_t(s) = \max_{\alpha}[R(\alpha, t) + \gamma \cdot U_{t+1}(succ(\alpha))] \tag{18}$$

where α is any action that can be taken from s and γ is a factor with $0 \leq \gamma \leq 1$ which provides a weighting for the relationship between the importance of rewards which are earned in the near and in the far future.

The policy $\Pi_t(s)$ defines the action α the robot should take when in state s at time t in order to maximize the reward:

$$\Pi_t(s) = \arg\max_{\alpha}[R(\alpha, t) + \gamma \cdot U_{t+1}(succ(\alpha))]. \tag{19}$$

We also define the aforementioned finite horizon $H = t_{max} \cdot \Delta t^{-1}$, which is the number of time steps Δt from the start of the planning process to t_{max}. The initial state s_0 is the state the robot is in when the plan is determined. Since the planning procedure will be called at the beginning of the search and whenever a person has been found, we can define $s_0 = s'_k$ with $\mathbb{R}_k$ being the region the robot is in.

Given the computed policy $\Pi_{t \ldots H-1}$, we can generate the plan $\mathcal{P}^*_{t,s} = \{\mathcal{P}^*(t), \mathcal{P}^*(t+1), \ldots, \mathcal{P}^*(H-1)\}$. This plan is a sequence of actions as shown in Algorithm 1.

Algorithm 1 $\mathcal{P}^*_{t,s}(\Pi_{t\dots H-1})$

$\mathcal{P}^*(t) = \Pi_t(s)$;
for $k \leftarrow t+1$ **to** $H-1$ **do**
  $\mathcal{P}^*(k) = \Pi_k(succ(\mathcal{P}^*(k-1)))$;
end for
return $\mathcal{P}^*$;

From the backwards induction approach, we know the plan $\mathcal{P}^*_{t+1,\alpha} = \mathcal{P}^*_{t+1,succ(\alpha)}(\Pi_{t+1\dots H-1})$ when choosing an action $\Pi_t(s) = \alpha$ at time step t. Since we want to decrease the reward of a *rest* or *search* action when the region has already been searched in this plan in order to avoid endless search loops, we introduce a factor $h(\mathcal{P}^*_{t+1,\alpha}, \alpha)$ which reduces the reward when $region(\alpha)$ has been searched in $\mathcal{P}^*_{t+1,\alpha}$. We define the resulting reward as:

$$R'(\alpha, t) = h(\mathcal{P}^*_{t+1,\alpha}, \alpha) \cdot R(\alpha, t). \tag{20}$$

The value $0 \leq h(\mathcal{P}^*_{t+1,\alpha}, \alpha) \leq 1$ depends on when and how often $region(\alpha)$ has been searched in this plan. This greedy approach is shown in Algorithm 2 which can be used to compute the entire plan $\mathcal{P}^*$.

Algorithm 2 M-USB Planning

Input: $R(\alpha, t)$, t_{max}, $\mathbb{S}$, $s_0 \in \mathbb{S}$, γ;
Output: Reward maximizing plan $\mathcal{P}^*$;
$H \leftarrow t_{max}/\Delta t$;
$U_H(s) \leftarrow 0\ \forall s \in \mathbb{S}$;
for $t \leftarrow H-1$ **to** 0 **do**
  $U_t(s) = \max_{\alpha}[R'(\alpha, t) + \gamma \cdot U_{t+1}(succ(\alpha))]$;
  $\Pi_t(s) = \arg\max_{\alpha}[R'(\alpha, t) + \gamma \cdot U_{t+1}(succ(\alpha))]$;
end for
return $\mathcal{P}^*_{0,s_0}(\Pi_{0\dots H-1})$;

3 Simulated Experiments

Simulation Setup

To test the performance of the M-USB Search, we use a simulator we have developed to simulate a robot in a realistic retirement home environment. The simulation was executed on a Ubuntu machine with an AMD A10-5700 Processor and 12GB RAM.

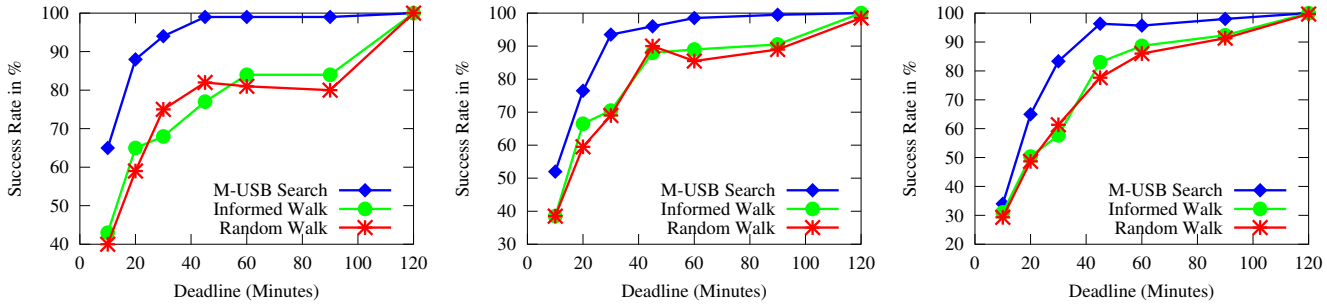
Simulation Environment. We created a map of a floor in a retirement home with 25 residents. The map consists of the residents' private rooms, two common rooms (TV-Room and Games Room), one Dining Hall, two Shower rooms, one Nurse Station, one Room for Family visits, and an outdoor Garden. All residents have their own unique schedules for the day. These schedules contain three meal times, breakfast (8 am-9 am), lunch (12 pm-1 pm), and dinner (6 pm-7 pm) during which the residents are available for the robot to interact with them. In addition, each schedule includes one 1-hour activity during which the residents are also available for interaction (e.g., walk and reading) and 2 to 4 appointments during which they must not be disturbed (e.g., doctor's visit). In his/her spare time, each resident visits random rooms at random times. A probability of $p_{miss} = 0.1$ is given for the residents not participating in their scheduled activities and behaving as if they have spare time. The simulated residents move with a speed of $v_p = 0.15\,\mathrm{m/s}$. The map used for these experiments is shown in Figure 1.

Performance Comparison. We compare the performance of our M-USB Search to both a *Weighted Informed Walk* and a *Random Walk* approach for the problem of a robot finding a group of residents within a deadline in the retirement home setting in order to remind them of an upcoming group-based recreational activity. The robot uses a speed of $v = 0.6\,\mathrm{m/s}$ and can detect people within a sensing range of $r = 1.8\,\mathrm{m}$ with respect to itself. If one of the searched residents is found, the robot stops for 1 minute in order to interact with the found person. The investigated search algorithms are:

1. **Random Walk.** The robot chooses a random room in the map, drives to this room and starts a local search in the room. This is repeated until all target residents are found or until the deadline is reached.
2. **Weighted Informed Walk.** Similar to the Random Walk, the Weighted Informed Walk algorithm picks a random room, drives there and starts a search in this room. However, a higher weighting is given to a resident's private room and common rooms. Namely, a weighting technique is applied to identify the importance of the regions accordingly to their room-classes. The weights for the different room-classes are: 0.5 for the room-class containing the private rooms of all searched residents; 0.25 for the room-class containing the common rooms; and 0.25 for a third room-class containing all other rooms. We assign an individual weight to each room in the environment. Namely, this individual weight is defined to be the corresponding weight for the room-class the room of interest is in divided by the number of rooms in this room-class. The robot applies a universal stochastic sampling technique based on the individual weights of the rooms to choose a room to search. The algorithm also considers the last 4 regions it has searched and does not search them again before 4 other regions have also been searched.
3. **M-USB-Search.** The proposed M-USB Search is used with a time discretization of $\Delta t = 10\,\mathrm{s}$. The schedule analyzer uses $\Delta t = 30\,\mathrm{s}$ and the database in which the robot saves the learned behaviour operates with $\Delta t = 300\,\mathrm{s}$. The attractivities for the actions are $\delta(\alpha_d) = 0.9 \cdot \delta(\alpha_s)$

Table 1: The $p_{\mathbb{C}}$ values used in the experiments.

Room Class	$p_{\mathbb{C}}$
Common Rooms	0.35
Corridors	0.05
Resident's private room ($\mathbb{C}^{p'}_{private}$)	0.4
Rooms in other room-classes	0.2



(a) N=5 (b) N=10 (c) N=15

Figure 5: Comparison Results: success rates for the different N.

Table 2: Comparison Results: mean search time per person, and number of visited regions within the environment for the different N.

Approach	Mean Search Time (Min.)			Visited Regions (mean)		
	N = 5	N = 10	N=15	N = 5	N = 10	N=15
M-USB Search	10.3	12.1	16.5	48	55	68
Weighted Informed Walk	15.5	18.1	23.2	176	194	202
Random Walk	14.6	15.9	24.0	119	137	214

and $\delta(\alpha_r) = 0.7 \cdot \delta(\alpha_s)$. For Eqs. (18) and (19), $\gamma = 0.99$ is used. For the learned behaviour we use $\alpha = 0.1$ in Eq. (6). In Eq. (9) the probabilities in Table 1 are applied to determine $\mathcal{L}_{env}(p, \mathbb{R}, t)$. To avoid endless search loops the robot needs to search 4 other rooms before searching the same room again. In particular, the value of $h(\mathcal{P}^*_{t+1,\alpha})$ for action α is set to zero when the room $region(\alpha)$ is contained in the next 4 searched regions in $\mathcal{P}^*_{t+1,\alpha}$.

Local Search in a Region. As our focus in this paper is on the high-level search to regions, for this comparison all aforementioned search approaches use the same random walk local search approach when they are searching within the region. The search time in the individual rooms is set to one second per squared meter.

The Search Queries. Each search approach was tested with different search queries q, which consisted of a robot finding $N = |\mathbb{P}_q|$ residents within different deadlines d. We used $N = 5, 10, 15$ and $d = 10, 20, 30, 45, 60, 90, 120$ minutes. For all combinations of N and d, we conducted 20 experiments. The robot started in the Games Room at 1:30 pm and searched for residents in $\mathbb{P}_q$ in order to invite them to a Bingo game that started at 4:00 pm. The start times were chosen such that the robot searched for residents in a time frame encompassing cases where residents had appointments, activities and spare time. The time t_0 indicates the time when the query was received.

Search Performance and Runtime

The performance metrics for the comparison are the success rate, the mean search time per person and the number of visited regions during the search procedure. We measure the pre-computation time needed at software start-up to create the MDP model and load the learned behaviour, and set up the three likelihood functions $\mathcal{L}_s$, $\mathcal{L}_l$, and $\mathcal{L}_{env}$. We also measure the computation times for the single plans (including the computation of $\mathcal{L}_{lkrl}$, the rewards, and the policies) which are computed when a query is received and whenever a new plan is generated due to replanning when a person has been found. The metrics are measured for: 1) different values of K, which represents the number of persons for which the plan is generated, and 2) for the different plan execution times, namely the time the robot plans into the future.

Results and Discussion

The comparison results are presented in Figure 5 and Table 2. Figure 5 shows that the proposed M-USB search finds more persons within a given deadline when compared to the other two approaches. All search algorithms have higher success rates for larger d since more residents can be found when more time is allocated to the search. It is interesting to note that the Weighted Informed Walk had comparable success rates to the Random Walk. We suspect that this is due to the time of the day in which the search took place. During portions of the search, some residents had activities in the garden. For the Weighted Informed Walk approach, the garden (which was not considered to be a common room) had a lower weight compared to the common rooms (TV-Room and Games Room) and private rooms. Therefore, these residents were not found in the majority of the searches with $d = 30, 45, 60, 90$ when using the Weighted Informed Walk approach due to the low weight given to the garden, which prevented the robot to visit this region often. However, these residents were found using the M-USB Search approach be-

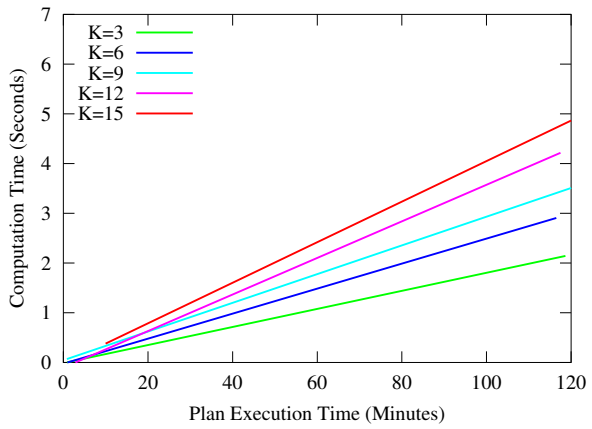


Figure 6: The M-USB Search computation time needed to compute a plan for K residents and a plan execution time.

cause the Schedule Analyzer increased the reward for the garden when one or more searched residents had an activity in this region.

The measured mean search times per person in Table 2 show that our M-USB Search is the fastest search approach. Furthermore, our M-USB Search approach finds a person by visiting the least amount of regions in the environment as also shown in Table 2. The overall results show that the use of the persons' schedules, learned behaviours, the topology of the environment, and the attempt to keep the search order in the proposed approach lowers the mean search times and the number of visited regions, and improves success rate.

The measured mean pre-computation time during system start-up for our approach was 40.51 s. The computation time needed when a query was received can be seen in Figure 6. It is linear for the number of residents (K) for whom the plan has to be generated since $\mathcal{L}_{lkrl}$ and the reward have to be computed K times. The computation time also increases with the plan execution time since the reward and the policy have to be computed for every time step during backwards induction. In general, the computation times are very short (e.g., we measure a mean of 4.8 s for $K = 15$ and a plan execution time of 120 minutes).

4 Conclusion

In this paper we address the problem where a robot needs to search for multiple non-static residents within a retirement home environment. We have developed the M-USB Search planning procedure which generates a high-level-plan to maximize the number of residents that are found within a given time frame. We obtain spatio-temporal likelihood functions for the individual residents using the schedules of the residents, the layout of the retirement home environment as well as direct observations by the robot. The M-USB search method uses a novel approach to compute the reward to determine the robot's search plan for finding multiple persons. We have compared our M-USB search method to a Weighted Informed Walk search and a Random Walk search for the proposed problem. Our results showed that the M-USB Search can find the residents in a shorter amount of time by visiting a fewer number of rooms.

Acknowledgments

This research has been funded by a Natural Sciences and Engineering Research Council of Canada (NSERC) Collaborative Research and Development Grant and by Dr Robot Inc. The first author has been funded by the German Academic Exchange Service (DAAD) PROMOS program.

References

Bath, P. A., and Deeg, D. 2005. Social engagement and health outcomes among older people: introduction to a special section. *European Journal of Ageing* 2(1):24–30.

Centre for Health Workforce Studies. 2006. The Impact of the Aging Population on the Health Workforce in the United States: Summary of Key Findings.

Elinas, P.; Hoey, J.; and Little, J. J. 2003. HOMER: Human Oriented MEssenger Robot. *AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments, Stanford CA*.

Findlay, R. A. 2003. Interventions to reduce social isolation amongst older people: where is the evidence? *Ageing and Society* 23(5):647–658.

Lau, H.; Huang, S.; and Dissanayake, G. 2005. Optimal search for multiple targets in a built environment. In *IROS*, 3740–3745. IEEE.

Lau, H.; Huang, S.; and Dissanayake, G. 2006. Probabilistic Search for a Moving Target in an Indoor Environment. In *IROS*, 3393–3398. IEEE.

McColl, D.; Louie, W.-Y. G.; and Nejat, G. 2013. Brian 2.1: A Socially Assistive Robot for the Elderly and Cognitively Impaired. *IEEE Robot. Automat. Mag.* 20(1):7483.

Menec, V. H. 2003. The relation between everyday activities and successful aging: A 6-year longitudinal study. *The Journals of Gerontology Series B: Psychological Sciences and Social Sciences* 58(2):S74–S82.

Oida, Y.; Kanoh, M.; Inagaki, M.; Konagaya, Y.; and Kimura, K. 2011. Development of a robot-assisted activity program for elderly people incorporating reading aloud and arithmetic calculation. In *Asian Perspectives and Evidence on Health Promotion and Education*. Springer. 67–77.

PriceWaterCoopers LLP. 2001. Report of a Study to Review Levels of Service and Responses to Need in a Sample of Ontario Long Term Care Facilities and Selected Comparators.

Sharkey, S. 2008. People caring for people impacting the quality of life and care of residents of long-term care homes: a report of the independent review of staffing and care standards for long-term care homes in ontario.

Tipaldi, G. D., and Arras, K. O. 2011. I want my coffee hot! Learning to find people under spatio-temporal constraints. In *ICRA*, 1217–1222. IEEE.

Wilson, R. S.; Krueger, K. R.; Arnold, S. E.; Schneider, J. A.; Kelly, J. F.; Barnes, L. L.; Tang, Y.; and Bennett, D. A. 2007. Loneliness and Risk of Alzheimer Disease. *Arch Gen Psychiatry* 64(2):234–240.

Position Paper: Synthesis of Plans for Robots from Plan Outlines

Srinivas Nedunuri and **Sailesh Prabhu** and **Mark Moll** and **Swarat Chaudhuri** and **Lydia E. Kavraki**
Dept. of Computer Science, Rice University, Houston, TX 77251
{nedunuri | snp3 | mmoll | swarat | kavraki}@rice.edu

Abstract

Planning the actions of a robot requires a combination of task planning and motion planning. Much of the programming today for doing this requires great attention to detail and careful coding in languages like C++ or Python. This is laborious, error-prone, and time consuming. We propose an alternative approach in which integrated task and motion plans are *synthesized* from a combination of programmer written statements, constraints on the solution, logical requirements, preferences, and pre-computed partial robot motions.

Motivation. For a long time most robots were confined to steel cages on factory floors with carefully controlled operating environments. Recently though, a new breed of robots is moving out of the confines of the cells in which they were formerly trapped to interact with their broader environment. Examples include the Kiva robots (Kiv) and hospital robots such as the RP-VITA (RPV). This implicitly requires the robot to have certain capabilities such as the ability to maintain global invariants, meet goals, and accept constraints on the resulting plans.

Unfortunately, the required tool support to be able to program such capabilities has not kept pace with the demands. For example, despite the abstraction provided by ROS (Quigley et al. 2009) and frameworks such as MoveIt! (Chitta, Şucan, and Cousins 2012), programming still involves non-trivial coding in languages such as C++. Not only is writing and modifying such programs error-prone but it also requires a lot of details to be supplied by the programmer. Considering the requirements on a tool to support the above capabilities, it is difficult to see how current approaches will scale.

Our Proposal. Integrated Task and Motion Planning (see e.g., (Nedunuri et al. 2014)) is a challenging class of planning problems involving a complex combination of high-level *task planning* and low-level *motion planning*. Task planning level is discrete and requires combinatorial search of the space of possible integrated plans, while motion planning is responsible for finding paths in continuous spaces. We propose to *synthesize* a correct integrated task and motion plan from a combination of programmer written statements, constraints on the solution, requirements, preferences, and pre-computed partial robot motions. The general goal of program synthesis is to semi-automatically construct a program that satisfies a given specification. Additionally, in template-based synthesis (Solar-Lezama et al. 2006), the programmer provides a template, or what we call a *plan outline*. The plan outline allows the programmer to supply known ordering and control information (these constitute the programmer written statements mentioned above). At the same time, it frees the programmer from having to specify information which the tool can infer, such as the specific paths to take, the exact location of objects, or the order in which to move them, conditions on branch statements, etc.

Although our approach is not specific to any problem domain, to convey the scope of the problem we are targeting, consider a warehouse where packed order boxes are brought to shipping and stacked on a table. Now consider a robot whose job it is to remove boxes from the table, load them onto dollies or pods, take the dollies out to the appropriate delivery vans, and return to the table with the empty dollies. The goal of the robot is to transport all boxes to the delivery vans while respecting any restrictions on the motion planning level, such as path lengths, clearances, areas to avoid, etc. Programming the robot for such a scenario requires integrated task and motion planning and determining the required conditions on the robot to ensure smooth operation.

Our Approach. To be able to synthesize an integrated plan, the following additional information will be needed in addition to a plan outline:

- a *scene description* that specifies the physical workspace in which the robot operates;
- a *robot model*, which specifies what actions the robot is capable of;
- a set of logical *requirements* that the generated plan must satisfy along with *preferences* on those requirements.

In the case of the warehouse example, the scene description supplies the layout of the warehouse, such as the location of floors, doors, wings, walls, etc. as well as demarcating certain non-permanent regions of interest, such as loading, maintenance, break areas, etc. The requirements allow the programmer to control the solution produced by the tool, e.g., that particular sections of the warehouse should be avoided, or certain types of deliveries such as perishables have priority and must be stocked within a given timeframe.

In our recent work (Nedunuri et al. 2014) we have developed a simple version of a tool which can handle small benchmark problems. The plan outline and requirements are written as a C-like "program" where the objects that the robot moves around (for example, boxes) are declared as symbolic variables, and actions that the robot performs (for example, moving or picking up an object) appear as function calls. From the scene description, the existing tool first constructs a finite graph, we call a *placement graph*, with nodes that represent appropriately chosen configurations of the robot base (base nodes) and nodes representing possible robot poses in which the robot can potentially grasp an object in a stable location (location nodes). Edges in the graph between base nodes represent motions that do not require the robot to move its arms, edges between base nodes and location nodes represent pick-and-place actions that are feasible without moving the base, and (directed) edges between location nodes indicate that an object in the location represented by the source node would block access to a location represented by the target node.

Next, using program analysis techniques, the tool automatically computes a logical formula that represents the set of all integrated plans that satisfy the structure of the plan outline and the requirements, and are also consistent with edges in the placement graph. The problem of finding a plan that meets all the criteria now resolves to computing a satisfying solution of this formula. This is done using Z3 (De Moura and Bjørner 2008), a state-of-the-art SMT solver. In a nutshell, SMT solvers are fully automatic, highly engineered programs that check the truth value of a logical formula containing symbols with a fixed interpretation, for example $\leq$ or $+$. The ability to express constraints such as $length(path1) + length(path2) < 100$ is the reason we use SMT solvers over plain Boolean satisfiability (SAT) solvers.

There are a number of very fundamental extensions to the existing tool which we propose to tackle, such as ongoing or recurring requirements (maintenance goals), reactivity, non-deterministic outcomes, goal preferences, enriching the constraint language, and dynamically constructing the placement graph. Our position is that the approach we outline, namely one of taking programmer supplied partial plans and constructing logical formulas which can be automatically solved to provide the missing information, is one that will scale to handle these sophisticated extensions.

Related Work. Our proposal is influenced by prior work in the programming language community on *template-based program synthesis* (Solar-Lezama et al. 2006; Srivastava, Gulwani, and Foster 2010). One key difference is that we are targeting robot plans that must be physically realizable.

Our approach differs from fully automated classical planning (Nau, Ghallab, and Traverso 2004) in that we accept a richer variety of inputs that influence the final solution. The idea of providing programmer input or domain specific knowledge in planning is of course not new. HTN (Hierarchical Task Network) planners (Nau, Ghallab, and Traverso 2004) share with our approach the input of domain knowledge. However, HTN require a level of domain expertise in order to correctly, completely, and efficiently codify the space of possible plans. In contrast, we believe it is important to accept partial user knowledge in a form that is familiar to programmers, and be able to provide a degree of flexibility in how much knowledge the programmer must provide: In general, the more the programmer provides, the better a solution that will be found (within the limits of the tool of course). Our proposal also extends to incorporate global constraints, which are difficult to handle in conventional automated (STRIPS-style) planners (Nareyek et al. 2005) .

A different form of hierarchy from that of HTN is used in Hierarchical Planning in the Now (HPN) (Kaelbling and Lozano-Pérez 2011b). In HPN, actions are abstracted by postponing of their preconditions, and the planning problem is solved using the abstract actions. The abstract solution is then refined into a concrete one. While HPN is very powerful (and has been extended to belief space planning (Kaelbling and Lozano-Pérez 2011a)), it rests on the assumption that planning problems can be decomposed into sub-problems that can be independently solved. While this is generally true, there are situations in which it can lead to dead-ends (e.g., if the robot exceeds the time limit for achieving some goal). Although HPN can always re-plan, there may be situations in which this can prove expensive (e.g., loading boxes onto a truck only to discover there is insufficient space for the last box).

Another major research effort has been the automatic synthesis of both reactive (Kress-Gazit, Fainekos, and Pappas 2009; Wongpiromsarn, Topcu, and Murray 2010) and non-reactive controllers (Guo, Johansson, and Dimarogonas 2013) and plans (Bhatia et al. 2011) for robots from temporal logic specifications. Tools such as LTLMOP (Kress-Gazit, Fainekos, and Pappas 2009) accept structured temporal specifications written in "natural language" style and handle reactive robotics. However, due to decidability limitations, the tool relies on a *propositional* representation of the problem. Besides the fact that we accept programmer information, the primary difference between these approaches and our proposal is that we use a *first-order* representation, where a space consisting of a vast number of plausible integrated plans can be represented concisely in the form a quantifier-free logical formula containing variables. Such a formula can be solved with a small number of calls to an automated solver. In contrast, methods relying on the explicit enumeration of a space may not scale as well, even when that space is symbolically represented using, for example, Binary Decision Diagrams (BDDs) (Bryant 1992).

Acknowledgments. Work on this paper by M.M. and L.K. was supported in part by NSF NRI 1317849 and 1139011 grants. Work by S.N., S.P., and S.C. was supported in part by NSF Award 1162076 and NSF CAREER Award 1156059.

References

Bhatia, A.; Maly, M.; Kavraki, L.; and Vardi, M. 2011. Motion planning with complex goals. *IEEE Robotics & Automation Magazine* 18(3):55 –64.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.* 24(3).

Chitta, S.; Şucan, I.; and Cousins, S. 2012. MoveIt! *IEEE Robotics & Automation Magazine* 19(1):18–19.

De Moura, L., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 337–340.

Guo, M.; Johansson, K.; and Dimarogonas, D. 2013. Motion and action planning under LTL specifications using navigation functions and action description language. In *Proc. of the Intl. Conf. on Intelligent Robots and Systems (IROS)*.

Kaelbling, L. P., and Lozano-Pérez, T. 2011a. Planning in the know: Hierarchical belief-space task and motion planning. In *Workshop on Mobile Manipulation, IEEE Intl. Conf. on Robotics and Automation*.

Kaelbling, L., and Lozano-Pérez, T. 2011b. Hierarchical task and motion planning in the now. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 1470–1477.

Kiva warehouse management system. http://www.kivasystems.com/resources/demo.

Kress-Gazit, H.; Fainekos, G.; and Pappas, G. 2009. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. on Robotics* 25(6):1370–1381.

Nareyek, A.; Freuder, E.; Fourer, R.; Giunchiglia, E.; Goldman, R.; Kautz, H.; Rintanen, J.; and Tate, A. 2005. Constraints and AI planning. *IEEE Intelligent Systems* 20(2):62–72.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann.

Nedunuri, S.; Prabhu, S.; Moll, M.; Chaudhuri, S.; and Kavraki, L. 2014. SMT-based synthesis of integrated task and motion plans from plan outline. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

RP-VITA. http://www.intouchhealth.com/products-and-services/products/rp-vita-robot/.

Solar-Lezama, A.; Tancau, L.; Bodik, R.; Seshia, S.; and Saraswat, V. 2006. Combinatorial sketching for finite programs. In *Proc. of the 12th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, 404–415.

Srivastava, S.; Gulwani, S.; and Foster, J. S. 2010. From program verification to program synthesis. In *Proc. 37th ACM Symp. Principles of Prog. Lang. (POPL)*, 313–326.

Wongpiromsarn, T.; Topcu, U.; and Murray, R. 2010. Receding horizon control for temporal logic specifications. In *Proc. of the 13th ACM Intl. Conf. on Hybrid Systems: Computation and Control*, 101–110.

Heuristic Search for Task and Motion Planning

Caelan Reed Garrett and **Tomás Lozano-Pérez** and **Leslie Pack Kaelbling**
MIT CSAIL

Abstract

Manipulation problems involving many objects present substantial challenges for motion planning algorithms due to the high dimensionality and multi-modality of the search space. Symbolic task planners can efficiently construct plans involving many entities but cannot incorporate the constraints from geometry and kinematics. In this paper, we show how to extend the heuristic ideas from one of the most successful symbolic planners in recent years, the FastForward (FF) planner, to motion planning, and to compute it efficiently. We use a multi-query roadmap structure that can be conditionalized to model different placements of movable objects. The resulting tightly integrated planner is simple and performs efficiently in a collection of tasks involving manipulation of many objects.

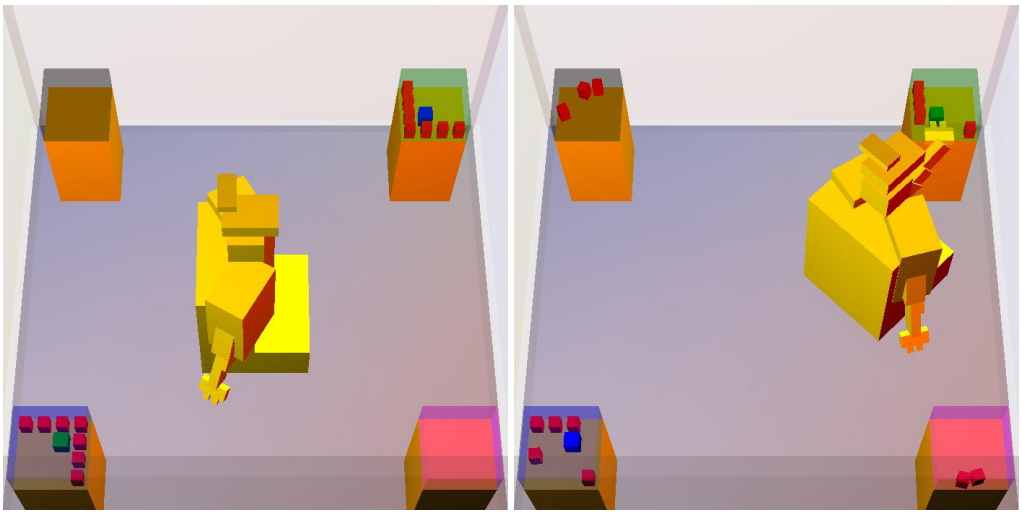
Introduction

Mobile manipulation robots are physically capable of solving complex problems involving moving many objects to achieve an ultimate goal. Mobile bases with one or more arms are becoming available and increasingly affordable while RGBD sensors are providing unprecedented sensory bandwidth and accuracy. However, these new capabilities are placing an increasing strain on existing methods for programming robots. Traditional motion-planning algorithms that find paths between fully specified configurations cannot address problems in which the configuration space of interest is not just that of the robot but the configuration space of a kitchen, for example, and the goal is to make dinner and clean the kitchen. We almost certainly do not want to choose whether to get the frying pan or the steak next by sampling configurations of the robot and kitchen and testing for paths between them.

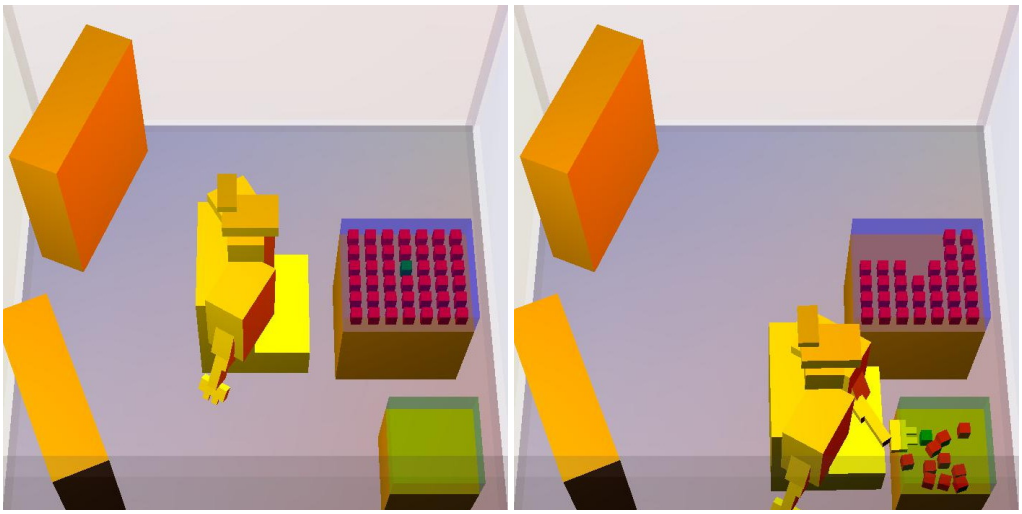
Researchers in artificial intelligence planning have been tackling problems that require long sequences of actions and large discrete state spaces and have had some notable success in recent years. However, these symbolic "task-level" planners do not naturally encompass the detailed geometric and kinematic considerations that motion planning requires. The original Shakey/STRIPS robot system (Fikes and Nilsson 1971; Nilsson 1984), from which many of these symbolic planners evolved, managed to plan for an actual robot by working in a domain where all legal symbolic plans were effectively executable. This required the ability to represent symbolically a sufficient set of conditions to guarantee the success of the steps in the plan. This is not generally possible in realistic manipulation domains because the geometrical and kinematic constraints are significant.

Consider a simple table-top manipulation domain where a variety of objects are placed on a table and the robot's task is to collect some subset of the objects and pack them in a box, or use them to make a meal, or put them away in their storage bins. The basic robot operations are to pick up an object and place it somewhere else; in addition, the robot can move its base in order to reach a distant object. Note that, in general, to reach some object, we will have to move other objects out of the way. Which objects need moving depends on their shapes, the shape of the robot, where the robot's base is placed and what path it follows to the object. When an object is moved, the choice of where to place it requires similar considerations. The key observation is that constructing a valid symbolic plan requires access to a characterization of the connectivity of the underlying free configuration space (for the robot and all the movable objects). We cannot efficiently maintain this connectivity with a set of static assertions updated by STRIPS operators; determining how the connectivity of the underlying free space changes requires geometric computation.

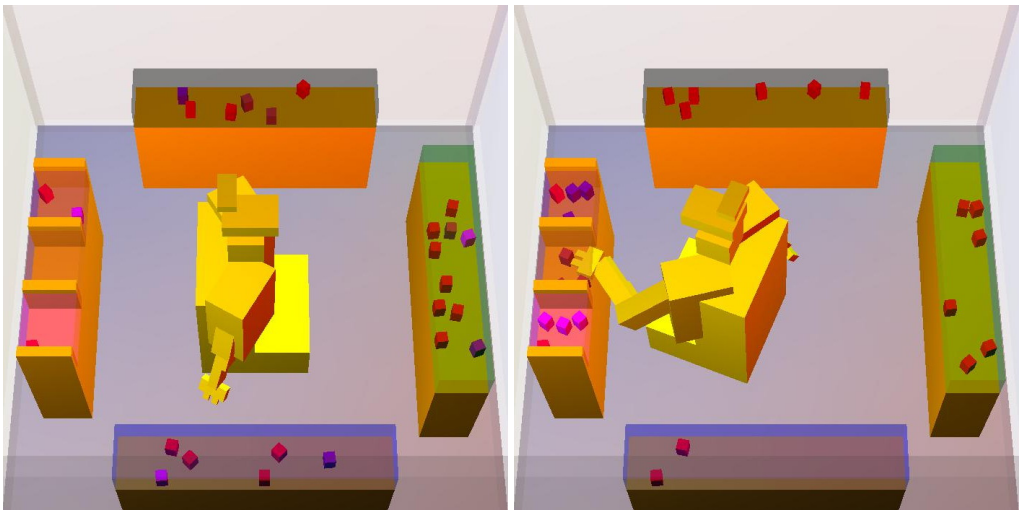
A natural extension to the classic symbolic planning paradigm is to introduce "computed predicates" (also know as "semantic attachments"); that is, predicates whose truth value is established not via assertion but by calling an external program that operates on a geometric representation of the state. A motion planner can serve to implement such a predicate, determining the reachability of one configuration from another. This approach is currently being pursued, for example, by Dornhege et al. (2009; 2013), as a way of combining symbolic task-level planners with motion planners to get a planner that can exploit the abstraction strengths of the first and the geometric strengths of the second. A difficulty with this approach, however, is that calling a motion planner is generally expensive. This leads to a desire to minimize the set of object placements considered, and, very importantly, to avoid calling the motion planner during heuristic evaluation. Considering only a sparse set of placements may limit the generality of the planner, while avoiding calling the motion planner in the heuristic leads to a heuristic that is uninformed about geometric considerations and may result in considerable inefficiency due to backtracking during the



(a) Median 18 actions



(b) Median 20 actions



(c) Median 32 actions

Figure 1: This figure illustrates the type of tasks addressed in this paper, involving a mobile manipulator (modeled on the PR2) that must move some set of specified objects (the ones not colored red) to specified target regions, moving the red objects as necessary to achieve this. Shown are the initial and final states in three of the tasks (numbered 3,4,5) used in our experiments.

search for a plan.

An alternative approach to integrating task and motion planning has been to start with a motion planner and use a symbolic planner to provide heuristic guidance to the motion planner, for example in the work of Cambon et al. (2009). However, since the task-level planner is ignoring geometry, its value as a heuristic is quite limited.

In this paper we show how to obtain a fully integrated task and motion planner using a search in which the heuristic takes geometric information into account. We show an extension of the heuristic used in the FastForward (FF) (Hoffmann and Nebel 2001) planning system to the FFRob heuristic, which integrates reachability in the robot configuration space with reachability in the symbolic state space. Both the search and the computation of the FFRob heuristic exploit a roadmap (Kavraki et al. 1996) data structure that allows multiple motion-planning queries on the closely related problems that arise during the search to be solved efficiently.

Related work

There have been a number of approaches to integrated task and motion planning in recent years. The pioneering Asymov system of Cambon et al. (2009) conducts an interleaved search at the symbolic and geometric levels. They carefully consider the consequences of using non-terminating probabilistic algorithms for the geometric planning, allocating computation time among the multiple geometric planning problems that are generated by the symbolic planner. The process can be viewed as using the task planner to guide the motion planning search. The work of Plaku and Hager (2010) is similar in approach.

The work of Erdem et al. (2011), is similar in approach to Dornhege et al. (2009), augmenting a task planner that is based on explicit causal reasoning with the ability to check for the existence of paths for the robot.

Pandey et al. (2012) and deSilva et al. (2013) use HTNs instead of generative task planning. Their system can backtrack over choices made by the geometric module, allowing more freedom to the geometric planning than in the approach of Dornhege et al. (2009). In addition, they use a cascaded approach to computing difficult applicability conditions: they first test quick-to-evaluate approximations of accessibility predicates, so that the planning is only attempted in situations in which it might plausibly succeed.

Lagriffoul et al. (2012) also integrate the symbolic and geometric search. They generate a set of approximate linear constraints imposed by the program under consideration, e.g., from grasp and placement choices, and use linear programming to compute a valid assignment or determine one does not exist. This method is particularly successful in domains such as stacking objects in which constraints from many steps of the plan affect geometric choices.

In the HPN approach of Kaelbling and Lozano-Pérez (2011), a regression-based symbolic planner uses *generators*, which perform fast approximate motion planning, to select geometric parameters, such as configurations and paths, for the actions. Reasoning backward using regression allows the goal to significantly bias the actions that are considered. This type of backward chaining to identify relevant actions is also present in work on *navigation among movable obstacles*. The work of Stilman et al. (2006; 2007) also plans backwards from the final goal and uses swept volumes to determine, recursively, which additional objects must be moved and to constrain the system from placing other objects into those volumes.

Srivastava et al. (2013; 2014) offer a novel control structure that avoids computing expensive precondition values in many cases by assuming a favorable default valuation of the precondition elements; if those default valuations prove to be erroneous, then it is discovered in the process of performing geometric planning to instantiate the associated geometric operator. In that case, symbolic planning is repeated. This approach requires the ability to diagnose why a motion plan is not possible in a given state, which can be challenging, in general. Empirically, their approach is the only one of which we are aware whose performance is competitive with our FFRob method.

All of these approaches, although they have varying degrees of integration of the symbolic and geometric planning, generally lack a true integrated heuristic that allows the geometric details to affect the focus of the symbolic planning. In this paper, we develop such a heuristic, provide methods for computing it efficiently, and show that it results in a significant computational savings.

Problem formulation

When we seek to apply the techniques of symbolic planning to domains that involve robot motions, object poses and grasps, we are confronted with a series of technical problems. In this section, we begin by discussing those problems and our solutions to them, and end with a formal problem specification.

We might naturally wish to encode robot operations that pick up and place objects in the style of traditional AI planning operator descriptions such as:

PICK(C_1, O, G, P, C_2):
pre: $HandEmpty$, $Pose(O, P)$,
$RobotConf(C_1)$, $CanGrasp(O, P, G, C_2)$,
$Reachable(C_1, C_2)$
add: $Holding(O, G)$, $RobotConf(C_2)$
delete: $HandEmpty$, $RobotConf(C_1)$

PLACE(C_1, O, G, P, C_2):
pre: $Holding(O, G)$,
$RobotConf(C_1)$, $CanGrasp(O, P, G, C_2)$,
$Reachable(C_1, C_2)$
add: $HandEmpty$, $Pose(O, P)$, $RobotConf(C_2)$
delete: $Holding(O, G)$, $RobotConf(C_1)$

In these operations, the C, P, and G variables range over robot configurations, object poses, and grasps, respectively. These are high-dimensional continuous quantities, which means that there are infinitely many possible instantiations of each of these operators. We address this problem by sampling finitely many values for each of these variable domains during a pre-processing phase. The sampling is problem-driven, but may turn out to be inadequate to support a solution. If this happens, it is possible to add samples and re-

attempt planning, although that was not done in the empirical results reported in this paper.

Even with finite domains for all the variables, there is a difficulty with explicitly listing all of the positive and negative effects of each operation. The operations of picking up or placing an object may affect a large number of $Reachable$ literals: picking up an object changes the "shape" of the robot and therefore what configurations it may move between; placing an object changes the free configuration space of the robot. Even more significant, which $Reachable$ literals are affected can depend on the poses of all the other objects (for example, removing any one or two of three obstacles may not render a configuration beyond the obstacles reachable). Encoding this conditional effect structure in typical form in the preconditions of the operators would essentially require us to write one operator description for each possible configuration of movable objects.

We address this problem by maintaining a state representation that consists of both a list of true literals and a data structure, called *details*, that captures the geometric state in a way that allows the truth value of any of those literals to be computed on demand. This is a version of the *semantic attachments* strategy (Dornhege et al. 2009).

The last difficulty is in computing the answers to queries in the details, especially about reachability, which requires finding free paths between robot configurations in the context of many different configurations of the objects. We address this problem by using a conditional *roadmap* data structure called a *conditional reachability graph*, related to a PRM (Kavraki et al. 1996), for answering all reachability queries, and lazily computing answers on demand and caching results to speed future queries.

More formally, a *state* is a tuple $\langle L, D\rangle$, where L is a set of literals and D is a domain-dependent detailed representation. A *literal* is a predicate applied to arguments, which may optionally have an attached *test*, which maps the arguments and state into a Boolean value. A literal *holds* in a state if it is explicitly represented in the state's literal set, or its test evaluates to true in the state:

$$\text{HOLDS}(l, s) \equiv l \in s.L \text{ or } l.test(s) \ .$$

A *goal* is a set of literals; a state *satisfies* a goal if all of the literals in the goal hold in the state:

$$\text{SATISFIES}(s, \Gamma) \equiv \forall l \in \Gamma.\ \text{HOLDS}(l, s) \ .$$

An *operator* is a tuple $\langle \phi, e_{pos}, e_{neg}, f\rangle$ where ϕ is a set of literals representing a conjunctive precondition, e_{pos} is a set of literals to be added to the resulting state, e_{neg} is a set of literals to be deleted from the resulting state, and f is a function that maps the detailed state from before the operator is executed to the detailed state afterwards. Thus, the successor of state s under operator a is defined

$$\text{SUCCESSOR}(s, a) \equiv \langle s.L \cup a.e_{pos} \setminus a.e_{neg}, a.f(s)\rangle \ .$$

An operator is *applicable* in a state if all of its preconditions hold in that state:

$$\text{APPLICABLE}(a, s) \equiv \forall l \in a.\phi.\ \text{HOLDS}(l, \phi) \ .$$

An *operator schema* is an operator with typed variables, standing for the set of operators arising from all instantiations of the variables over the appropriate type domains.

Our general formulation has broader applicability, but in this paper we restrict our attention to a concrete domain in which a mobile-manipulation robot can move, grasp rigid objects, and place them on a surface (see Figure 1). To formalize this domain, we use literals of the following forms:

- $RobotConf(C)$: the robot is in configuration C, where C is a specification of the pose of the base as well as joint angles of the arm;
- $Pose(O, P)$: object O is at pose P, where P is a four-dimensional pose (x, y, z, θ), assuming that the object is resting on a stable face on a horizontal surface;
- $Holding(O, G)$: the robot is holding object O with grasp G, where G specifies a transform between the robot's hand and the object;
- $HandEmpty$: the robot is not holding any object;
- $In(O, R)$: the object O is placed in such a way that it is completely contained in a region of space R; and
- $Reachable(C_1, C_2)$: there is a collision-free path between robot configurations C_1 and C_2, considering the positions of all fixed and movable objects as well as any object the robot might be holding and the grasp in which it is held.

The details of a state consist of the configuration of the robot, the poses of all the objects, and what object is being held in what grasp.

Two of these literals have tests. The first, In, has a simple geometric test, to see if object O, at the pose specified in this state, is completely contained in region R. The test for $Reachable$ is more difficult to compute; it will be the subject of the next section.

Conditional reachability graph

In the mobile manipulation domain, the details contain a *conditional reachability graph* (CRG), which is a partial representation of the connectivity of the space of sampled configurations, conditioned on the placements of movable objects as well as on what is in the robot's hand. It is similar in spirit to the roadmaps of Leven and Hutchinson (2002) in that it is designed to support solving multiple motion-planning queries in closely related environments. The CRG has three components:

- **Poses**: For each object o, a set of possible stable poses.
- **Nodes**: A set of robot configurations, c_i, each annotated with a (possibly empty) set $\{\langle g, o, p\rangle\}$ where g is a grasp, o an object, and p a pose, meaning that if the robot is at the configuration c_i, and object o is at pose p, then the robot's hand will be related to the object by the transform associated with grasp g.
- **Edges**: A set of pairs of nodes, with configurations c_1 and c_2, annotated with an initially empty set of *validation* conditions of the form $\langle h, g, o, p, b\rangle$, where b is a Boolean value that is TRUE if the robot moving from c_1 to c_2 along a simple path (using linear interpolation or some

other fixed interpolator) while holding object h in grasp g *will not* collide with object o if it is placed at pose p, and FALSE otherwise.

The validation conditions on the edges are not precomputed; they will be computed lazily, on demand, and cached in this data structure. Note that some of the collision-checking to compute the annotations can be shared, e.g. the same robot base location may be used for multiple configurations and grasps.

Constructing the CRG The CRG is initialized in a preprocessing phase, which concentrates on obtaining a useful set of sampled object poses and robot configurations. Object poses are useful if they are initial poses, or satisfy a goal condition, or provide places to put objects out of the way. Robot configurations are useful if they allow objects, when placed in useful poses, to be grasped (and thus either picked from or placed at those poses) or if they enable connections to other useful poses via direct paths. We assume that the following components are specified: a workspace W, which is a volume of space that the robot must remain inside; a placement region T, which is a set of static planar surfaces upon which objects may be placed (such as tables and floor, but not (for now) the tops of other objects); a set $\mathcal{O}_f$ of fixed (immovable) objects; a set $\mathcal{O}_m$ of movable objects; and a vector of parameters θ that specify the size of the CRG. It depends, in addition, on the start state s and goal Γ. We assume that each object $o \in \mathcal{O}_m$ has been annotated with a set of feasible grasps. The parameter vector consists of a number n_p of desired sample poses per object (type); a number n_{ik} of grasp configurations per grasp; a number n_n of configurations near each grasp configuration; a number n_c of RRT iterations for connecting configurations, and a number k specifying a desired degree of connectivity.

The CONSTRUCTCRG procedure is outlined below.

CONSTRUCTCRG$(W, T, s, \Gamma, \mathcal{O}_f, \mathcal{O}_m, \theta)$:

```
1  N = {s.details.robotConf} ∪ {robot configuration in Γ}
2  for o ∈ O_m:
3      P_o = {s.details.pose(o)} ∪ {pose of o in Γ}
4      for i ∈ {1, ..., θ.n_p}:
5          P_o.add(SAMPLEOBJPOSE(o.shape, T))
6          for g ∈ o.grasps:
7              for j ∈ {1, ..., θ.n_ik}:
8                  N.add(SAMPLEIK(g, o, p), (g, o, p))
9              for j ∈ {1, ..., θ.n_n}:
10                 N.add(SAMPLECONFNEAR(g, ( )))
11 E = { }
12 for n_1 ∈ N:
13     for n_2 ∈ NEARESTNEIGHBORS(n_1, k, N):
14         if CFREEPATH(n_1.c, n_2.c, O_f): E.add(n_1, n_2)
15 N, E = CONNECTTREES(N, E, W, θ.n_c)
16 return ⟨P, N, E⟩
```

We begin by initializing the set of nodes N to contain the initial robot configuration and the configuration specified in the goal, if any. Then, for each object, we generate a set of sample poses, including its initial pose and goal pose, if any, as well as poses sampled on the object placement surfaces. For each object pose and possible grasp of the object, we use the SAMPLEIK procedure to sample one or more robot configurations that satisfy the kinematic constraints that the object be grasped. We sample additional configurations with the hand near the grasp configuration to aid maneuvering among the objects. We then add edges between the k nearest neighbors of each configuration, if a path generated by linear interpolation or another simple fixed interpolator is free of collisions with fixed objects. At this point we generally have a forest of trees of configurations. Finally, we attempt to connect the trees using an RRT algorithm as in the sampling-based roadmap of trees (Plaku et al. 2005).

To test whether this set of poses and configurations is plausible, we use it to compute a heuristic value of the starting state, as described in section . If it is infinite, meaning that the goal is unreachable even under extremely optimistic assumptions, then we return to this procedure and draw a new set of samples.

Querying the CRG Now that we have a CRG we can use it to compute the test for the *Reachable* literal, as shown in REACHABLETEST below.

REACHABLETEST$(c_1, c_2, D, \text{CRG})$:

```
1  for (o, p) ∈ D.objects:
2      for e ∈ CRG.E:
3          if not ⟨D.heldObj, D.grasp, o, p, *⟩ ∈ e.valid:
4              p = CFREEPATH(e.n_1.c, e.n_2.c, o@p,
5                            D.heldObj, D.grasp)
6              e.valid.add(⟨D.heldObj, D.grasp, o, p,
7                           (p != None)⟩)
8  G = {e ∈ CRG.E | ∀(o, p) ∈ D.objects.
9           ⟨D.heldObj, D.grasp, o, p, True⟩ ∈ e.valid}
10 return REACHABLEINGRAPH(c_1, c_2, G)
```

The main part of the test is in lines 8–10: we construct a subgraph of the CRG that consists only of the edges that are valid given the object that the robot is holding and the current placements of the movable objects and search in that graph to see if configuration c_2 is reachable from c_1. Lines 1–7 check to be sure that the relevant validity conditions have been computed and computes them if they have not. The procedure CFREEPATH$(c_1, c_2, obst, o, g)$ performs collision checking on a straight-line, or other simply interpolated path, between configurations c_1 and c_2, with a single obstacle $obst$ and object o held in grasp g.

In addition, the CRG is used to implement APPLICABLEOPS(s, Ω, CRG), which efficiently determines which operator schema instances in Ω are applicable in a given state s. For each schema, we begin by binding variables that have preconditions specifying the robot configuration, object poses, the currently grasped object and/or the grasp to their values in state s. We consider all bindings of variables referring to objects that are not being grasped. For a *pick* operation, P is specified in the current state, so we consider all bindings of G and C_2 such that $(C_2, (G, O, P)) \in$ CRG.N. For a *place* operation, G is specified in the current state, so we consider all bindings of P and C_2 such that $(C_2, (G, O, P)) \in$ CRG.N.

Planning algorithms

A *planning problem*, Π, is specified by $\langle s, \Gamma, \mathcal{O}, T, W, \Omega \rangle$, where s is the initial state, including literals and details, Γ is the goal, $\mathcal{O}$ is a set of objects, T is a set of placement surfaces, W is the workspace volume, and Ω is a set of operator schemas.

PLAN, shown below, is a generic heuristic search procedure. Depending on the behavior of the EXTRACT procedure, it can implement any standard search control structure, including depth-first, breadth-first, uniform cost, best-first, A^*, and hill-climbing. Critical to many of these strategies is a heuristic function, which maps a state in the search to an estimate of the cost to reach a goal state from that state. Many modern domain-independent search heuristics are based on a *relaxed plan graph* (RPG). In the following section, we show how to use the CRG to compute the relaxed plan graph efficiently.

PLAN(Π, EXTRACT, HEURISTIC, θ)

```
1  ⟨s, Γ, O, T, W, Ω⟩ = Π
2  CRG = CONSTRUCTCRG(W, T, s, Γ, O, θ)
3  def H(s): HEURISTIC(RPG(s, Γ, CRG, Ω))
4  q = QUEUE(SEARCHNODE(s, 0, H(s), None))
5  while not q.empty():
6      n = EXTRACT(q)
7      if SATISFIES(n.s, Γ): return n.path
8      for a ∈ APPLICABLEOPS(n.s, Ω, CRG):
9          s' = SUCCESSOR(n.s, a)
10         q.push(SEARCHNODE(s', n.cost + 1, H(s'), n))
```

Computing the relaxed plan graph In classical symbolic planning, a *plan graph* is a sequence of alternating *layers* of literals and actions. The first layer consists of all literals that are true in the starting state. Action layer i contains all operators whose preconditions are present and simultaneously achievable in literal layer i. Literal layer $i + 1$ contains all literals that are possibly achievable after i actions, together with a network of *mutual exclusion* relations that indicates in which combinations those literals might possibly be true. This graph is the basis for *GraphPlan* (Blum and Furst 1997) and related planning algorithms.

The *relaxed plan graph* is a simplified plan graph, without mutual exclusion conditions; it is constructed by ignoring the negative effects of the actions. From the RPG, many heuristics can be computed. For example, the H_{Add} heuristic (Bonet and Geffner 2001) returns the sum of the levels at which each of the literals in the goal appears. It is optimistic, in the sense that if the mutual exclusion conditions were taken into account, it might take more steps to achieve each individual goal from the starting state; it is also pessimistic, in the sense that the actions necessary to achieve multiple goal fluents might be "shared." An admissible heuristic, H_{Max} (Bonet and Geffner 2001), is obtained by taking the maximum of the levels of the goal literals, rather than the sum; but it is found in practice to offer weaker guidance. An alternative is the FF heuristic (Hoffmann and Nebel 2001), which performs an efficient backward-chaining pass in the plan graph to determine how many actions, if they could be performed in parallel without deletions, would be necessary to achieve the goal and uses that as the heuristic value. An important advantage of the FF heuristic is that it does not over-count actions if one action achieves multiple effects, and it enables additional heuristic strategies that are based on *helpful actions*. We use a version of the helpful-action strategy that reduces the choice of the next action to those that are in the first level of the relaxed plan, and find that it improves search performance.

In order to use heuristics derived from the RPG we have to show how it can be efficiently computed when the add lists of the operators are incomplete and the truth values of some literals are computed from the CRG in the details. We present a method for computing the RPG that is specialized for mobile manipulation problems. It constitutes a further relaxation of the RPG which allows literals to appear earlier in the structure than they would in an RPG for a traditional symbolic domain. This is necessary, because the highly conditional effects of actions on $Reachable$ literals makes them intractable to compute exactly. The consequence of the further relaxation is that the H_{Add} and H_{Max} heuristics computed from this structure have less heuristic force. However, in section we describe a method for computing a version of H_{FF} that recovers the effectiveness of the original.

The intuition behind this computation is that, as we move forward in computing the plan graph, we consider the positive results of all possible actions to be available. In terms of reachability, we are removing geometric constraints from the details; we do so by removing an object from the universe when it is first picked up and never putting it back, and by assuming the hand remains empty (if it was not already) after the first *place* action. Recall that, in APPLICABLE and SATISFIES, the HOLDS procedure is used to see if a literal is true in a state. It first tests to see if it is contained in the literal set of the state; this set becomes increasingly larger as the RPG is computed. If the literal is not there, then it is tested with respect to the CRG in the details, which becomes increasingly less constrained as objects are removed.

Importantly, since the geometric tests on the CRG are cached, the worst-case number of geometric tests for planning with and without the heuristic is the same. In practice, computing the RPG for the heuristic is quite fast, and using it substantially reduces the number of states that need to be explored.

RELAXEDPLANGRAPH, shown below, outlines the algorithm in more detail.

RELAXEDPLANGRAPH$(s, \Gamma, \text{CRG}, \Omega)$:

1 $D = s.D$; $ops = $ ALLNONCONFBINDINGS(Ω)
2 $literals = [\,]$; $actions = [\,]$; $hState = s$
3 **while True**
4 $layerActions = \{\,\}$; $layerLiterals = \{\,\}$
5 **for** $op \in ops$:
6 **if** APPLICABLE$(op, hState)$:
7 $layerActions.add(op)$
8 $layerLiterals.union(op.e_{pos})$
9 $ops.remove(op)$
10 **if** $op.type = pick$: $D.objects.remove(op.obj)$
11 **if** $op.type = place$: $D.heldObj =$ **None**
12 $literals.append(layerLiterals)$
13 $actions.append(layerActions)$
14 $hState = \langle \bigcup_i literals_i, D \rangle$
15 **if** SATISFIES$(hState, \Gamma)$: **return** $(literals, actions)$
16 **if** $layerActions = \{\,\}$: **return None**

In the second part of line 1, in a standard implementation we would generate all possible instantiations of all actions. However, because of the special properties of reachability, we are able to abstract away from the particular configuration the robot is in when an action occurs; thus, we consider all possible bindings of the non-configuration variables in each operator, but we only consider binding the starting configuration variable to the actual current starting configuration and leave the resulting configuration variable free. In line 2, we initialize $hState$, which is a pseudo-state containing all literals that are possibly true at the layer we are operating on, and a set of details that specifies which objects remain as constraints on the robot's motion at this layer. In line 6, we ask whether a operator schema with all but the resulting configuration variable bound is applicable in the heuristic state. We only seek a single resulting configuration that satisfies the preconditions of op in $hState$; even though many such configurations might exist, each of them will ultimately affect the resulting $hState$ in the same way. Lines 7–9 constitute the standard computation of the RPG. In lines 10–11 we perform domain-specific updates to the detailed world model: if there is any way to pick up an object, then we assume it is completely removed from the domain for the rest of the computation of the RPG; if there is any way to put down the currently held object, then we assume that there is no object in the hand, when doing any further computations of reachability in the CRG. Line 14 creates a new $hState$, which consists of all literals possibly achievable up to this level and the details with possibly more objects removed.

There is one last consideration: the strategy shown above does not make the dependencies of $Reachable$ literals at level i on actions from level $i-1$ explicit; the truth of those literals is encoded implicitly in the details of the $hState$. We employ a simple bookkeeping strategy to maintain a causal connection between actions and literals, which will enable a modified version of the FF heuristic to perform the backward pass to find a parallel plan. We observe that, in the relaxed plan, once an object is $pick$ed, it is effectively removed from the domain. So, we add an extra positive effect literal, $Picked(o)$ to the positive effects set of the $pick$ action, just when it is used in the heuristic computation.

The FFRob heuristic The FF heuristic operates by extracting a *relaxed plan* from the RPG and returning the number of actions it contains. A relaxed plan $\mathcal{P}$ constructed for starting state s and set of goal literals G consists of a set of actions that has the following properties: (1) For each literal $l \in G$ there is an action $a \in \mathcal{P}$ such that $l \in a.e_{pos}$ and (2) For each action $a \in \mathcal{P}$ and each literal $l \in a.\phi$, either $l \in s$ or there exists an action $a' \in \mathcal{P}$ such that $l \in a.e_{pos}$.

That is, the set of actions in the relaxed plan collectively achieve the goal as well as all of the preconditions of the actions in the set that are not satisfied in the initial state. It would be ideal to find the shortest linear plan that satisfied these conditions, however that is NP-hard (Hoffmann and Nebel 2001). Instead, the plan extraction procedure works backwards, starting with the set of literals in the goal G. For each literal $l \in G$, it seeks the "cheapest" action a^* that can achieve it; that is,

$$a^* = \arg \min_{\{a | l \in a.e_{pos}\}} \sum_{l \in a.\phi} \mathcal{L}(l) \ ,$$

where $\mathcal{L}(l)$ is the index of the lowest layer containing l (which is itself a quick estimate of the difficulty of achieving l.)

The minimizing a^* is added to the relaxed plan, l and any other literals achieved by a^* are removed from the goal set, and the preconditions $a^*.\phi$ are added to the goal set unless they are contained in s. This process continues until the goal set is empty.

The RPG computed as in section does not immediately support this computation, because the $Picked$ fluents that are positive results of $Pick$ actions do not match the $Reachable$ fluents that appear in preconditions. In general, there may be many ways to render a robot configuration reachable, by removing different combinations of obstacles. Determining the smallest such set is known as the *minimum constraint removal* problem (Hauser 2014). Hauser shows it is NP-Hard in the discrete case and provides a greedy algorithm that is optimal if obstacles must not be entered more than once. We have extended this method to handle the case in which objects are *weighted*; in our case, by the level in the RPG at which they can be picked. The weighted MCR algorithm attempts to find a set of obstacles with a minimal sum of weights that makes a configuration reachable.

So, any action precondition of the form $Reachable(c)$ is replaced by the set of preconditions $Picked(o)$ for all objects o in the solution to the weighted MCR problem for configuration c. This represents the (approximately) least cost way to make c accessible. Having carried out this step, we can use the standard FF method for extracting a relaxed plan. The FFRob heuristic returns the number of actions in this relaxed plan.

Geometric biases It frequently happens that multiple states have the same heuristic value; in such cases, we break ties using geometric biases. These three biases do not affect the overall correctness or completeness of the algorithm. Intuitively, the idea is to select actions that maximize the reachability of configurations in the domain from the current state.

T	Pre	No H			H_{FF}			H_{AddR}			H_{FFR}, HA			H_{FFRB}			H_{FFRB}, HA		
		t	m	s	t	m	s	t	m	s	t	m	s	t	m	s	t	m	s
0	21	265	35	48719	102	72	6123	41	19	536	6	5	78	7	5	87	2	0	23
1	25	300	0	63407	283	17	14300	162	55	2042	3	0	8	16	11	153	4	1	49
2	29	300	0	50903	300	0	8947	300	0	3052	5	1	12	17	13	114	7	2	32
3	23	300	0	39509	300	0	4849	300	0	1767	83	19	464	99	43	523	13	1	69
4	30	300	0	23920	300	0	1574	300	0	1028	300	0	1274	18	3	20	16	3	20
5	51	300	0	9422	300	0	1533	300	0	592	300	1	272	106	17	32	99	14	32

Figure 2: The results of running the algorithms on each of the six tasks (T). Each entry in the table reports *median time (t) in seconds* (shown in gray), *median absolute deviation, MAD, of the times (m)*, and *median states (s) expanded*. Each task also incurs a pre-processing time (Pre, in seconds) for building the roadmap.

- Choose actions that leave the largest number of configurations corresponding to placements of objects in their goal poses or regions available. This captures the idea that blocking goal regions should be avoided if possible. This is useful because although a heuristic will report when a placement is immediately bad, i.e., already blocking future goals, it will not convey information that the placement may prevent two necessary placements later in the search because it was out in the open. This is because the relaxed plan assumed that a free placement exists, despite objects being placed there, because it does not model negative effects of actions.
- Choose actions that leave the largest total number of configurations corresponding to placements reachable; this ensures that all placements are as tight as possible against the edge of the reachable space.
- If neither of the previous biases breaks the tie, then select actions that maximize the total number of reachable configurations.

These biases experimentally prove to be helpful in giving the search additional guidance in this domain, especially in combination with enforced hill climbing search, which lacks backtracking to undo bad decisions.

Results

We have experimented with various versions of this algorithm, differing in the definition of the heuristic, on a variety of tasks; we report the results in this section.

The search strategy in all of our experiments is enforced hill-climbing (Hoffmann and Nebel 2001), in which a single path through the state space is explored, always moving to the unvisited successor state with the smallest heuristic value, with ties broken using geometric biases. This search strategy is known not to be complete, but we have found it to be very effective in our domains. If the hill-climbing search were to reach a dead end, one could restart the search (as is done in FastForward), using the best-first strategy or weighted A^*, which are complete. However, even with a complete search and no helpful-action heuristic, the overall planner is not probabilistically complete, since it is limited to the initial set of sample poses and configurations.

The parameters governing the creation of the CRG are: $n_p \in [25 - 50]$ (the number of placements for each object); this varies with the size of the placement regions; $n_{ik} = 1$ (number of robot configurations for each grasp); $n_n = 1$ (number of additional robot configurations near each grasp); $n_c = 250$ (number of RRT iterations); $k = 4$ (number of nearest neighbors).

In our experiments, we generate an initial CRG using these parameters during pre-processing and then test whether the value of the heuristic at the initial state is finite. If it is not, we discard it and try again, with the same parameters. Very few retries were necessary to find a CRG with finite heuristic value. This condition was effective: in every case in our experiments, the CRG contained a valid plan.

The following versions of the planner are compared in the experiments:

1. No H: The heuristic always returns 0.
2. H_{FF}: This is the original heuristic in FF, based only on the symbolic literals, completely ignoring the reachability conditions when computing the heuristic. Helpful actions are not used.
3. H_{AddR}: This is a version of the original H_{Add} heuristic that returns the sum of the levels of the RPG at which the goal literals are first found. This makes use of the CRG to reason about reachability. It does not build a relaxed plan and, therefore, does not have helpful actions.
4. $H_{\mathrm{FFR,HA}}$: This computes the RPG, does a backward scan to find a relaxed plan and computes helpful actions based on that plan.
5. H_{FFRB}: Like H_{FFR} but using geometric biases to break ties and without using helpful actions.
6. $H_{\mathrm{FFRB,HA}}$: Like H_{FFR} but using geometric biases to break ties and using helpful actions.

We tested our algorithm on 6 different *tasks*, in which the goals were conjunctions of $In(O_i, R_j)$ for some subset of the objects (the ones not colored red). Other objects were moved as necessary to achieve these goals. The last three tasks are shown in Figure 1; the first three are tasks are simpler variations on task 3 (Figure 1(a)).

The table in Figure 2 shows the results of running the algorithms in each of the tasks. Each entry in the table reports *median time (t) in seconds* (shown in gray), *median absolute deviation, MAD, of the times (m)*, and *median states (s) expanded*. Each task also incurs a pre-processing time for building the roadmap; this is reported (in seconds) in the Pre column of the table. The median-based robust statistics

are used instead of the usual mean and standard deviation since the data has outliers. Entries with a median time of 300 and MAD of 0 did not successfully complete any of the simulations. There were 20 simulations per task for the first two heuristics and 120 simulations per task for the others. Running times are from a Python implementation running on a 2.6GHz Intel Core i7.

As can be clearly seen, especially in the number of expanded states, exploiting geometric information in the heuristic produces substantial improvements. Introducing geometric biases to settle ties helps in the most cluttered of the examples.

Conclusion We have shown how to combine data structures for multi-query motion planning algorithms with the search and heuristic ideas from the FF planning system to produce a deeply integrated task and motion planning system. The integrated heuristic in this system is quite effective in focusing the search based on geometric information at relatively low cost.

Acknowledgements This work was supported in part by the NSF under Grant No. 1117325. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from ONR MURI grant N00014-09-1-1051, from AFOSR grant FA2386-10-1-4135 and from the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center.

References

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artif. Intell.* 90(1-2):281–300.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* 28.

de Silva, L.; Pandey, A. K.; Gharbi, M.; and Alami, R. 2013. Towards combining HTN planning and geometric task planning. In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic attachments for domain-independent planning systems. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 114–121. AAAI Press.

Dornhege, C.; Hertle, A.; and Nebel, B. 2013. Lazy evaluation and subsumption caching for search-based integrated task and motion planning. In *IROS workshop on AI-based robotics*.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Hauser, K. 2014. The minimum constraint removal problem with three robotics applications. *I. J. Robotic Res.* 33(1):5–17.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal Artificial Intelligence Research (JAIR)* 14:253–302.

Kaelbling, L. P., and Lozano-Perez, T. 2011. Hierarchical planning in the now. In *IEEE Conference on Robotics and Automation (ICRA)*.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

Lagriffoul, F.; Dimitrov, D.; Saffiotti, A.; and Karlsson, L. 2012. Constraint propagation on interval bounds for dealing with geometric backtracking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Leven, P., and Hutchinson, S. 2002. A framework for real-time path planning in changing environments. *I. J. Robotic Res.* 21(12):999–1030.

Nilsson, N. J. 1984. Shakey the robot. Technical Report 323, Artificial Intelligence Center, SRI International, Menlo Park, California.

Pandey, A. K.; Saut, J.-P.; Sidobre, D.; and Alami, R. 2012. Towards planning human-robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement. In *RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*.

Plaku, E., and Hager, G. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Plaku, E.; Bekris, K. E.; Chen, B. Y.; Ladd, A. M.; and Kavraki, L. E. 2005. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics* 21(4):597–608.

Srivastava, S.; Riano, L.; Russell, S.; and Abbeel, P. 2013. Using classical planners for tasks with continuous operators in robotics. In *ICAPS Workshop on Planning and Robotics (PlanRob)*.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE Conference on Robotics and Automation (ICRA)*.

Stilman, M., and Kuffner, J. J. 2006. Planning among movable obstacles with artificial constraints. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*.

Stilman, M.; Schamburek, J.-U.; Kuffner, J. J.; and Asfour, T. 2007. Manipulation planning among movable obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Extending Knowledge-Level Contingent Planning for Robot Task Planning

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Andre Gaschler
fortiss GmbH
An-Institut Technische Universität München
Munich, Germany
gaschler@fortiss.org

Abstract

We present a set of extensions to the knowledge-level PKS (Planning with Knowledge and Sensing) planner, aimed at improving its ability to generate plans in real-world robotics domains. These extensions include a facility for integrating externally-defined reasoning processes in PKS (e.g., invoking a motion planner), an interval-based fluent representation for capturing the effects of noisy sensors and effectors, and an application programming interface (API) to facilitate software integration on robot platforms. We demonstrate our techniques in three simple robot domains, which show their applicability to a broad range of robot planning applications involving incomplete knowledge, real-world geometry, and multiple robots and sensors.

Introduction and Motivation

A robot operating in a real-world domain often needs to do so with *incomplete information* about the state of the world. A robot with the ability to *sense* the world can also gather information to generate plans with *contingencies*, allowing it to reason about the outcome of sensed data at plan time.

In this paper, we explore an application of planning with incomplete information and sensing actions to the problem of task planning in robotics domains. In particular, building models of realistic domains which can be used with general-purpose planning systems often involves working with incomplete (or uncertain) perceptual information arising from real-world sensors. Furthermore, this task may be complicated by the difficulties of bridging the gap between geometric and symbolic representations: robot systems typically reason about joint angles, spatial coordinates, and continuous spaces, while many symbolic planners work with discrete representations in represented in logic-like languages.

Our approach makes use of the PKS (Planning with Knowledge and Sensing) planner (Petrick and Bacchus 2002; 2004) as the high-level reasoning tool for task planning in robotics domains. PKS is a general-purpose contingent planner that operates at the *knowledge level* (Newell 1982), by reasoning about how its knowledge changes due to action during plan generation. PKS is able to represent known and unknown information, and model sensing actions using concise but rich domain descriptions, making it well suited for reasoning in structured, partially-known environments of the kind that arise in many robot scenarios.

Figure 1: In the FORCE SENSING scenario, a compliant robot manipulator senses if beverage containers are filled by lifting them and sensing their weight. Objects must be held upright while moving to prevent spilling, unless they are known to be completely empty or unopened.

While PKS has been used successfully in previous robot domains (Petrick et al. 2009), it lacks certain features which could improve its applicability to a wider range of robotics tasks. In this paper, we describe a set of extensions designed to improve PKS's ability to generate plans in real-world robot scenarios, by focusing on three tasks: combining high-level symbolic planning with low-level motion planning, reasoning about noisy sensors and effectors, and facilitating planner-level software integration on robot platforms.

The planner has also been integrated into a larger software framework called *Knowledge of Volumes for robot task Planning (KVP)* (Gaschler et al. 2013a), aimed at facilitating the use of planning techniques on a variety of robot platforms (see Figures 1 and 5), which has been developed as part of the JAMES project.[1] This framework serves as the basis for the robot demonstrators we describe below.

The rest of this paper is organised as follows. We first

[1] See http://james-project.eu/ for more information.

present an overview of PKS and then describe three extensions to the basic planning system which enhance its ability to operate in robotics domains. We then give three examples of robot domains where we use the extended version of the planner to generate solutions; the first two domains are tested on real robots, while the third domain is tested in simulation. Finally, we situate our approach with respect to related research and discuss future directions of our work.

Planning with Knowledge and Sensing (PKS)

PKS (Planning with Knowledge and Sensing) is a contingent planner that builds plans in the presence of incomplete information and sensing actions (Petrick and Bacchus 2002; 2004). PKS works at the *knowledge level* by reasoning about how the planner's knowledge state, rather than the world state, changes due to action. PKS works with a restricted subset of a first-order logical language, and limited inference. Thus, unlike planners that reason directly with possible worlds models or belief states, PKS works with a set of formulae representing the planner's knowledge state. This enables it to support a rich representation with features such as functions and variables; however, as a trade-off, its restricted representation means that the planner cannot model certain types of knowledge.

PKS is based on a generalisation of STRIPS (Fikes and Nilsson 1971). In STRIPS, the state of the world is modelled by a single database. Actions update this database and, by doing so, update the planner's world model. In PKS, the planner's knowledge state, rather than the world state, is represented by a set of five databases, each of which models a particular type of knowledge. The contents of these databases have a fixed, formal interpretation in a modal logic of knowledge. Actions can modify any of the databases, which has the effect of updating the planner's knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) that it can represent. The contents of the databases are as follows:

$\mathbf{K_f}$: This database is like a STRIPS database except that both positive and negative facts are permitted and the closed world assumption is not applied. K_f is used for modelling action effects that change the world. K_f can include any ground literal ℓ, where $\ell \in K_f$ means "the planner knows ℓ." K_f can also contain known function (in)equality mappings.

$\mathbf{K_w}$: This database models the plan-time effects of sensing actions with binary outcomes. $\phi \in K_w$ means that at plan time the planner either "knows ϕ or knows $\neg\phi$," and that at execution time this disjunction will be resolved. In such cases we will also say that the planner "knows whether ϕ." Know-whether information is important since PKS uses such knowledge to construct conditional plans (see below).

$\mathbf{K_v}$: This database stores information about function values that will become known at execution time. In particular, K_v can model the plan-time effects of sensing actions that return constants, such as numeric values. K_v can contain any unnested function term f, where $f \in K_v$ means that at plan time the planner "knows the value of f." At execution time, the planner will have definite information about f's value. As a result, PKS is able to use K_v terms as *run-time variables* (Etzioni et al. 1992) in its plans.

$\mathbf{K_x}$: This database models the planner's *exclusive-or* knowledge. Entries in K_x have the form $(\ell_1|\ell_2|\ldots|\ell_n)$, where each ℓ_i is a ground literal. Such formulae represent a particular type of disjunctive knowledge that arises in many planning scenarios, namely that "exactly one of the ℓ_i is true."

$\mathbf{LCW}$: This database stores the planner's *local closed world* information (Etzioni, Golden, and Weld 1994), i.e., instances where the planner has complete information about the state of the world. We will not use LCW in this paper.

PKS's databases can be inspected through a set of *primitive queries* that ask simple questions about the planner's knowledge state. Simple knowledge assertions can be tested with a query $K(\phi)$ which asks: "is a formula ϕ true?" A query $K_w(\phi)$ asks whether ϕ is known to be true or known to be false (i.e., does the planner "know whether ϕ"). A query $K_v(t)$ asks "is the value of function t known?" The negation of the above queries can also be used. An inference procedure is used to evaluate primitive queries by checking the contents of the databases, taking into consideration the interaction between different types of knowledge.

An action in PKS is modelled by a set of *preconditions* that query the agent's knowledge state, and a set of *effects* that update the state. Action preconditions are simply a list of primitive queries. Action effects are described by a collection of STRIPS-style "add" and "delete" operations that modify the contents of individual databases. E.g., $\mathit{add}(K_f, \phi)$ adds ϕ to K_f, and $\mathit{del}(K_w, \phi)$ removes ϕ from K_w. Actions can also have ADL-style context-dependent effects (Pednault 1989), where the secondary preconditions of an effect are described by lists of primitive queries. A simple form of quantification, $\forall^K x$ and $\exists^K x$, that ranges over known instantiations of x can also be used. Examples of PKS actions are shown below in Figure 3.

PKS constructs plans by reasoning about actions in a simple forward-chaining manner: if the preconditions of an action are satisfied by the planner's knowledge state, then the action's effects are applied to produce a new knowledge state. Planning then continues from the resulting state. PKS can also build contingent plans with branches, by considering the possible outcomes of its K_w and K_v knowledge. For instance, if $\phi \in K_w$ then PKS can construct two conditional branches in a plan: along one branch (the K^+ branch) ϕ is assumed to be known (i.e., ϕ is added to K_f), while along the other branch (the K^- branch), $\neg\phi$ is assumed to be known (i.e., $\neg\phi$ is added to K_f). A similar type of multi-way branching plan can also be built by considering a restricted type of K_v information. Planning continues along each branch until the *goal*—a list of primitive queries–is satisfied. A sample plan with branches is shown in Figure 4, and described in greater detail below.

Extensions to PKS for Robot Task Planning

In this section we consider three recent extensions to the basic PKS system which we believe are particularly useful for robot task planning. First, we describe a mechanism which

allows externally-defined procedures (e.g., from support libraries) to be integrated with the internal reasoning mechanisms of the planner. Second, we present an extension of the PKS representation which allows a form of noisy numerical information to be modelled, for instance to represent the effects of error prone sensors. Finally, we describe a software-level application programming interface to PKS, which aids in the engineering task of integrating the planner with a robot system.

Executing Externally-Defined Procedures

The first extension we describe aims to take advantage of existing reasoning tools by providing a mechanism for PKS to invoke externally-defined procedures (e.g., defined in special purpose libraries) from within the planner's internal reasoning mechanism during plan generation. While this idea is not new, and has been successfully applied in other contexts (see the discussion section, below), the introduction of this technique into PKS is a recent extension to the planner.

In particular, PKS provides an external query mechanism of the form:

$$\texttt{extern}(\mathit{proc}(\vec{x})),$$

where `extern` is a special keyword indicating that control should be transferred to an external procedure with the name *proc*. $\vec{x}$ is a set of parameters that should be passed to *proc*. In general, x can contain any symbols defined in PKS's knowledge state, providing a link between the planner and the externally-defined procedure. An `extern` call can be used within an action definition, either as a precondition or an effect. The return value of the `extern` call, defined within the external procedure, is passed back to PKS, which interprets it in the context where it occurs in the action. Additional tests may be performed on this value, which can be assigned to domain properties and included in the planner's knowledge state. While no restrictions are placed on when such procedures can be used in a planning domain, in practice `extern` calls are most useful if used for complex or special purpose reasoning that cannot easily be modelled in the planner's restricted representation language, or where more efficient reasoning engines already exist.

The `extern` mechanism provides a powerful tool for PKS to use in robotics domains by augmenting PKS's core reasoning capabilities with the addition of motion planning, collision detection, and other special purpose robotics libraries. For instance, geometric predicates and continuous motions can be evaluated with `extern` calls, and reasoned about at the symbolic level, enabling us to solve problem instances which may be difficult to model directly at either the motion planning or symbolic planning level alone. Examples of this process are given below.

One important drawback with this facility in its present form, is that there is no control over how long an external procedure may take to execute, or whether it will terminate at all. As a result, we are currently extending our `extern` implementation to introduce a simple timeout facility that will force external procedure calls to terminate if a specified cutoff time is reached. Currently, however, the domain designer must ensure that any externally-defined procedures operate correctly in the context of a given planning domain.

Reasoning with Interval-Valued Fluents

One type of sensed information that arises in many real-world robotics contexts is *numerical* information, which is often necessary for modelling state properties (e.g., the robot is 10 metres from the wall), limited resources (e.g., ensure the robot has enough fuel), constraints (e.g., only grasp an object if its radius is less than 10 cm), or arithmetic operations (e.g., advancing the robot one step reduces its distance to the wall by 1 metre). Reasoning with incomplete numerical information is often problematic, however, especially when planners represent incompletely known state properties by sets of states, each of which denotes a possible configuration of the actual world state. E.g., if a fluent f could map to any natural number between 1 and 100, then we require 100 states to capture f's possible mappings. The state explosion resulting from large sets of mappings can be computationally difficult for planners that must reason directly with individual states to construct plans.

In PKS, we build on a previous planning approach (Petrick 2011) which uses *interval-valued fluents (IVFs)* (Funge 1998) to avoid some of the computational problems involved with uncertain numerical information. The idea is simple: instead of representing each possible mapping by a separate state, a single interval mapping is used, where the endpoints of the interval indicate the fluent's range of possible values. Thus, a fluent f that could map to values between 1 and 100 can be denoted in an interval-valued form by $f = \langle 1, 100\rangle$.

In general, PKS treats each IVF as a function whose denotation is an *interval* of the form $\langle u, v\rangle$. The *endpoints* of the interval, u and v, indicate the bounds on the range of possible mappings for the fluent. Since we are interested in planning with incomplete information, a mapping $f = \langle u, v\rangle$ will mean that the value of f is known to be in the interval $\langle u, v\rangle$. If a fluent maps to a *point interval* of the form $\langle u, u\rangle$, then the mapping is certain and known to be equal to u.

PKS's knowledge of (general) IVFs are stored in its K_x database, as a generalisation of its exclusive-or information. In addition to basic intervals, *disjunctive intervals* (i.e., sets of disjoint interval mappings) are also permitted. For instance, if a fluent f could possibly map to any value between 5 and 10 or, alternatively, map to values between 15 and 18, we can represent such information by the K_x formula $(f = \langle 5, 10\rangle \,|\, f = \langle 15, 18\rangle)$.

Certain types of IVFs can also be represented in the K_v and K_w databases. For instance, a fluent of the form $f : \langle x - c, x + c\rangle$ in K_v means that the value of the fluent f is known, and f is in the range $x \pm c$, for some numeric constant c and unknown fluent value x. This mechanism can be used to model the results of noisy sensors. In K_w, we also permit numeric relations of the form f **op** c, where **op** $\in \{=, \neq, >, <, \geq, \leq\}$ and c is a numeric constant. Thus, $f > 5 \in K_w$ can be used to model a sensing action that determines whether f is greater than 5 or not. Since K_w is used to build contingent branches into a plan, this extension also enables PKS to build branches based on IVFs.

An Application Programming Interface

The task of integrating a planner onto a robot platform often centres around the problem of representation, and how

to abstract the capabilities of a robot and its working environment so that it can be put in a suitable form for use by the planner. Integration also typically requires the ability to communicate information between system components. Thus, the integration of a planning system usually requires a consideration of certain engineering-level concerns, to ensure proper interoperability with components that aren't traditionally considered in theoretical planning settings.

In order to facilitate the task of providing software-level planning services to robot systems, we have created an application programming interface (API) for a version of PKS implemented as a C++ library. This interface abstracts many common planning operations into a series of functions which provide direct access to these services. For instance, this interface includes methods for manipulating domain representations, as well as functions for controlling certain aspects of the the plan generation process itself (e.g., selecting goals, generation strategies, or planner-specific settings). Moreover, functions that allow plans to be manipulated as first-class entities (e.g., for replanning) are provided. A fragment of the API is given in Figure 2.

Overall, the API is designed to be generic and is not meant to be tied to one particular planning system. For instance, the planner configuration methods are meant to provide a way to set certain properties of the underlying planning system, and provide access to features needed for debugging. The domain configuration functions provide the main methods for defining planning domain models, either from traditional domain/problem files, or via string-based descriptions. One important idea behind the configuration functions is that they offer the possibility of specifying domains to the planner incrementally, using function calls alone, rather than specifying a single monolithic domain file. This means that an initial domain could be specified and then later revised, for instance due to additional information discovered by the robot during execution (e.g., new domain objects, revised action descriptions, additional properties corresponding to new capabilities of the robot, etc.). Finally, the plan generation and iteration functions specify methods for controlling various aspects of the plan generation process, and provide a way for processes external to the planner to control simple monitoring and replanning activities, including updates to certain aspects of the planning problem, such as goal change.

We will discuss the integration of PKS on our robot platforms in greater detail in the discussion section below.

```
// Configuration and debugging
void    reset();
string  getPlannerProperty(string);
bool    setPlannerProperty(string, string);

// Domain configuration
bool    defineDomain(string);
bool    defineSymbols(string);
bool    defineActions(string);
bool    defineProblems(string);
bool    definePlanState(string);
bool    defineObservedState(string);

// Plan generation and iteration
bool    buildPlan();
string  getCurrentPlan();
Action  getNextAction();
bool    isNextActionEndOfPlan();
bool    isPlanDefined();
bool    setProblem(string);
bool    setProblemGoal(string);
```

Figure 2: A fragment of the PKS API.

Example Domains

To demonstrate our approach, we now describe three robotics scenarios that make use of knowledge-level planning: the FORCE SENSING and the BIMANUAL robot scenarios, based on domains first described in (Gaschler et al. 2013c) and tested on real robots, and the ROBOT LOCALISATION scenario, tested in simulation. In all scenarios, the robot uses sensing actions to obtain knowledge of some domain property which is necessary for achieving the goal. In the first scenario, only the basic PKS system is used. In the second scenario, PKS's external procedure mechanism is used to link a motion planning library to the planner's internal reasoning mechanisms. In the final scenario, we make use of interval-valued fluents in a simple localisation task. In each case, we discuss the symbolic domain definitions of the scenario, and provide an example of the solution plan that was generated in that domain.

Force Sensing Scenario

In the FORCE SENSING scenario, a robot manipulator is tasked with transferring beverage containers from one table to another, as shown in Figure 1. Through its torque sensors, it can sense the external force of a grasped container, and decide whether or not that drink could be spilled. The robot should hold drinks exactly upright to prevent spilling, unless a drink is known to be completely empty, in which case a faster arbitrary motion may be performed. In order to keep this scenario simple, the location of all objects are known and no sensing except force sensing is available.

Figure 3 shows the PKS actions in the FORCE SENSING scenario, which includes a sensing action, `senseWeight`, which senses the weight of a beverage container `?o`. To perform this action, the robot must first be grasping object `?o`. To ensure only new knowledge is gained from this action, and to increase planning efficiency, we include a precondition that the robot must not yet know whether `?o` is spillable. When this action is performed, knowledge of whether `?o` is spillable or not is added to PKS's K_w database.

This scenario also includes a number of actions for manipulating domain objects, including `transferUpright`, `transfer`, `grasp`, and `ungrasp` actions, also listed in Figure 3. For example, in the `transferUpright` action, the robot can move a grasped container from one table to the other, while keeping the orientation of its parallel gripper fixed. Only objects that are grasped and not yet removed to the second table can be transferred.

An example plan for the FORCE SENSING scenario is shown in Figure 4 for the case of two objects in the domain. In particular, a sensing action is performed on each

```
action senseWeight(?o:object)
    preconds:
        ¬K_w(isSpillable(?o)) &
        K(isGrasped(?o))
    effects:
        add(K_w, isSpillable(?o))

action transfer(?o:object)
    preconds:
        K(¬isSpillable(?o)) &
        K(isGrasped(?o)) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isRemoved(?o))

action transferUpright(?o:object)
    preconds:
        K(isSpillable(?o)) &
        K(isGrasped(?o)) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isRemoved(?o))

action grasp(?o:object)
    preconds:
        K(emptyGripper) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isGrasped(?o)),
        add(K_f, ¬emptyGripper)

action ungrasp(?o:object)
    preconds:
        K(isGrasped(?o)) &
        K(isRemoved(?o))
    effects:
        add(K_f, ¬isGrasped(?o)),
        add(K_f, emptyGripper)
```

Figure 3: Actions in the FORCE SENSING domain.

object (`can1` and `can2`) and the objects are individually manipulated depending on whether their contents are spillable or not. The resulting plan therefore considers four contingent situations which could arise during plan execution. This scenario was tested on a joint-impedance controlled lightweight 7-DoF robot with a force-controlled parallel gripper. Forces were measured by internal torque sensing.

Bimanual Robot Scenario

The second scenario is a demonstration of a BIMANUAL robot (Figure 5) whose hands can reach different areas of a table. In this case, the robot can sense if bottles on the table are empty or full using a top-down camera. Its goal is to clean up all empty bottles by removing them to a certain "dishwasher" location. In order to achieve this goal, the robot must move objects that are only accessible by its left arm to a location that its right arm can reach, a behaviour which arises purely from symbolic planning. In contrast to the previous FORCE SENSING scenario, the BIMANUAL robot scenario relies on visual information, which can be gathered without requiring manipulation.

```
 1.  grasp(can1) ;
 2.  senseWeight(can1) ;
 3.  branch(isSpillable(can1))
 4.  K+:
 5.      transferUpright(can1) ;
 6.      ungrasp(can1) ;
 7.      grasp(can2) ;
 8.      senseWeight(can2) ;
 9.      branch(isSpillable(can2))
10.      K+:
11.          transferUpright(can2) ;
12.          ungrasp(can2).
13.      K-:
14.          transfer(can2) ;
15.          ungrasp(can2).
16.  K-:
17.      transfer(can1) ;
18.      ungrasp(can1) ;
19.      grasp(can2) ;
20.      senseWeight(can2) ;
21.      branch(isSpillable(can2))
22.      K+:
23.          transferUpright(can2) ;
24.          ungrasp(can2).
25.      K-:
26.          transfer(can2) ;
27.          ungrasp(can2).
```

Figure 4: A plan for removing 2 objects from a table in the FORCE SENSING domain.

The PKS actions in the BIMANUAL scenario are given in Figure 6. Two robot arms are tasked with removing all empty bottles that are visible on a table, and moving them to the dishwasher location, which can only be reached by the right robot arm. The domain includes one sensing action, `senseIfEmpty`, which has no precondition other than the requirement that the knowledge it gathers must be new. For manipulation, both robot arms can perform the `pickUp` and `putDown` actions. However, not all locations can be reached by both hands, so the preconditions of these actions include an **extern** call to `isReachable`, which is defined in a motion planning library and which checks reachability for a specific manipulator and location. This interaction of symbolic and motion planners is described in greater detail in the discussion section below, and in (Gaschler et al. 2013a).

An example plan is shown in Figure 7 for the case of 4 objects. In particular, the plan senses each object to detect whether or not it is empty and then constructs a conditional plan to subsequently remove the empty objects to the dishwasher. The resulting plan therefore considers 16 possible configurations of empty/non-empty bottles which could arise at execution. (The actions for the case where `bottle0` and `bottle2` are empty are shown.) It is interesting to observe that this simple robot scenario already gives rise to interesting behaviour: since the right arm cannot directly reach all objects that need to be transferred to the goal location, the left arm must pass those objects to a location reachable by both hands. This behaviour has not been pre-programmed, but instead arises purely from the combination of symbolic and geometric planning.

Figure 5: In the BIMANUAL scenario, a camera is used to recognise empty bottles which a bimanual robot should remove from the table to a "dishwasher" location on the left side, behind the table (Gaschler et al. 2013a; Giuliani et al. 2013). A video of the robot operating in this scenario is available at `http://youtu.be/yMmZkhHr8ss`.

This domain was tested on a two 6-DoF industrial manipulator setup with Meka Robotics H2 humanoid hands, with an RGB camera facing top-down for simple colour-filtering object recognition, as described in (Foster et al. 2012).

Robot Localisation Scenario

In the final example, we consider a robot whose location, represented by the IVF `robotLoc`, is measured by the robot's distance to a wall. The robot has two physical actions available to it: `moveForward`, which moves the robot either 1 or 2 units towards the wall; and `moveBackward`, which moves the robot 1 unit away from the wall. The robot also has a sensing action, `atTarget`, which senses whether the robot is at a target location, specified by the function `targetLoc`. Additionally, the robot also has a second sensing action, `withinTarget`, that determines whether or not the robot is within the target distance `targetLoc`.

The definitions of the PKS actions for this scenario are given in Figure 8 (all action preconditions are assumed to be true). The robot's initial location is specified by the interval mapping $\texttt{robotLoc} = \langle 3, 4\rangle$ stored in K_x. The goal is to move the robot to the target location, i.e., $K(\texttt{robotLoc} = \texttt{targetLoc})$, where $\texttt{targetLoc} = 2$ is stored in K_f.

One solution generated by PKS is the conditional plan in Figure 9. Since forward movements may change the robot's position by either 1 unit or 2 units, `noisyForward` in step 1 results in an even less certain position for the robot, namely $\texttt{robotLoc} = \langle 1, 3\rangle \in K_x$. However, the sensing action in step 2, together with the branch point in step 3, lets us split this interval into two parts. In step 4, we assume that $\texttt{robotLoc} \leq 2$ and consider the case where $\texttt{robotLoc} = \langle 1, 2\rangle$. `atTarget`, together with the branch in step 6, lets us divide this interval even further: in step 7, $\texttt{robotLoc} = 2$ and the goal is satisfied, while in step 8, $\texttt{robotLoc} = 1$ and a `moveBackward` action achieves the goal. In step 10 we consider the other sub-interval of the

```
action senseIfEmpty(?o:object)
    preconds:
        ¬K_w(isEmptyBottle(?o))
    effects:
        add(K_w, isEmptyBottle(?o))

action pickUp(?r:robot, ?o:object, ?l:location)
    preconds:
        K(?l = getObjectLocation(?o)) &
        K(handEmpty(?r)) &
        K(extern(isReachable(?l, ?r)))
    effects:
        del(K_f, ?l = getObjectLocation(?o)),
        del(K_f, handEmpty(?r)),
        add(K_f, inHand(?o, ?r))

action putDown(?r:robot, ?o:object, ?l:location)
    preconds:
        K(inHand(?o, ?r)) &
        K(extern(isReachable(?l, ?r)))
    effects:
        del(K_f, inHand(?o, ?r)),
        add(K_f, ?l = getObjectLocation(?o)),
        add(K_f, handEmpty(?r))
```

Figure 6: Actions in the BIMANUAL domain.

first branch, i.e., $\texttt{robotLoc} = 3 \in K_f$. In this case we have definite knowledge, however, a subsequent `noisyForward` results in $\texttt{robotLoc} = \langle 1, 2\rangle$. The remainder of the plan in steps 12–16 is the same as in steps 5–9: the robot conditionally moves backwards in the case that `robotLoc` is determined to be 1, while the plan trivially achieves the goal if $\texttt{robotLoc} = 2$.

We have not tested this domain on a real robot yet but have instead performed a series of tests in simulation using a variety of initial and target locations. Experimentation with IVF domains on a real robot is a focus of current work.

Related Work and Discussion

Applications of automated planning to robotics go back to the early 1980s, for instance with the famous robot systems Shakey (Nilsson 1984) and Handey (Lozano-Pérez et al. 1989). Since that time, the field has made substantial progress, and various approaches to robot task planning have been proposed, including probabilistic techniques from artificial intelligence (Kaelbling and Lozano-Pérez 2013), closed-world symbolic planning (Cambon, Alami, and Gravot 2009; Plaku and Hager 2010; Dornhege et al. 2009b), formal synthesis (Kress-Gazit and Pappas 2008; Cheng et al. 2012), and sampling-based manipulation planning (Zacharias, Borst, and Hirzinger 2006; Barry 2013).

As part of our work to apply general-purpose planning in robotics domains, we developed the *Knowledge of Volumes framework for robot task Planning (KVP)*, initially presented in (Gaschler et al. 2013a). KVP uses PKS as its underlying symbolic planner, and combines it with the idea of treating 3D geometric volumes as an intermediary representation between continuously-valued robot motions and discrete symbolic actions, to address the problem of bridg-

```
 1.  senseIfEmpty(bottle0) ;
 2.  senseIfEmpty(bottle1) ;
 3.  senseIfEmpty(bottle2) ;
 4.  senseIfEmpty(bottle3) ;
 5.  branch(isEmptyBottle(bottle0))
 6.  K+:
 7.    branch(isEmptyBottle(bottle1))
 8.    K+: ...
 9.    K-:
10.      branch(isEmptyBottle(bottle2))
11.      K+:
12.        branch(isEmptyBottle(bottle3))
13.        K+: ...
14.        K-:
15.            pickUp(left,bottle0,l0) ;
16.            putDown(left,bottle0,l5) ;
17.            pickUp(right,bottle2,l2) ;
18.            putDown(right,bottle2,dishwasher) ;
19.            pickUp(right,bottle0,l5) ;
20.            putDown(right,bottle0,dishwasher).
21.      K-: ...
22.  K-: ...
```

Figure 7: A plan for 4 objects in the BIMANUAL domain.

```
action moveForward
    effects:
        add(K_f, robotLoc := robotLoc - <1,2>)

action moveBackward
    effects:
        add(K_f, robotLoc := robotLoc + 1)

action atTarget
    effects:
        add(K_w, robotLoc = targetLoc)

action withinTarget
    effects:
        add(K_w, robotLoc <= targetLoc)
```

Figure 8: Actions in the LOCALISATION domain.

ing the gap between geometric and symbolic planning representations. By using the intermediate representation of volumes, KVP can model continuous geometry, in contrast to arbitrary discretisation (Gaschler et al. 2013a).

Previous work described the KVP framework (Gaschler et al. 2013a), and gave details of the swept volume computation for convex sets of polyhedra (Gaschler et al. 2013b). The two task planning scenarios discussed in this paper were previously presented in (Gaschler et al. 2013c), however, the present paper focuses on the planning aspects of this work, giving a detailed discussion of knowledge-level planning, sensing actions, and discrete uncertainty.

A number of approaches also address the problem of integrating symbolic planning and motion planning. For instance, our work is in part inspired by Kaelbling and Lozano-Pérez's earlier work on hierarchical task and motion planning (Kaelbling and Lozano-Pérez 2011), borrowing the continuous geometry of swept volumes. However, while the

```
                                          robotLoc
 0.                                       ⟨3, 4⟩
 1.  noisyForward ;                       ⟨1, 3⟩
 2.  withinTarget ;
 3.  branch(robotLoc ≤ targetLoc)
 4.  K+:                                  ⟨1, 2⟩
 5.      atTarget ;
 6.      branch(robotLoc = targetLoc)
 7.      K+: nop.                           2
 8.      K-:                                1
 9.          moveBackward.                  2
10.  K-:                                    3
11.      noisyForward ;                     ⟨1, 2⟩
12.      atTarget ;
13.      branch(robotLoc = targetLoc)
14.      K+: nop.                           2
15.      K-:                                1
16.          moveBackward.                  2
```

Figure 9: A plan in the LOCALISATION domain.

geometric preconditions may be similar, their underlying aggressively hierarchical planning strategy differs from the knowledge-level planning approach we use here. Further approaches that integrate symbolic and geometric reasoning are presented by Cambon, Alami and Gravot (2009), handling geometric preconditions and effects; Dornhege et al. (2009b); and, more recently, Plaku and Hager (2010), which additionally allow differential motion constraints in a sampling-based motion and action planner. We note that the latter three approaches assume a closed world, where all symbols must be either true or false. Our approach instead represents open-world knowledge, which allows us to model incomplete information and high-level sensing. Prior work has also used PKS to connect robot vision and grasping with automated planning (Petrick et al. 2009).

In terms of our extensions to PKS, the ability to link external libraries to internal reasoning processes is key to our approach. While this idea is not new, and has been previously applied (Eiter et al. 2006; Dornhege et al. 2009a; Erdem et al. 2011), the introduction of such techniques to PKS is a recent addition to the planner. Current work is focused on extending this interface, to allow external procedures partial access to internal PKS planning states, for more efficient external execution during plan generation.

Interval-valued numeric models have been previously investigated in planning contexts, e.g., for modelling time as a resource (Edelkamp 2002; Frank and Jónsson 2003; Laborie 2003). A similar representation to ours for bounding noisy numeric properties has also been proposed by Poggioni, Milani, and Baioletti (2003). This idea also has parallels to work on register models (van Eijck 2013). The importance of numerical reasoning in planning has been recognised with the inclusion of numeric state variables in PDDL, and in planners like MetricFF. We believe representations such as our IVF approach offer a useful middle ground between discrete and fully probabilistic models of uncertainty.

Motion planning and collision detection in our work rely heavily on the Robotics Library (RL)[2] (Rickert 2011), ex-

[2]Available from `http://www.roboticslibrary.org/`.

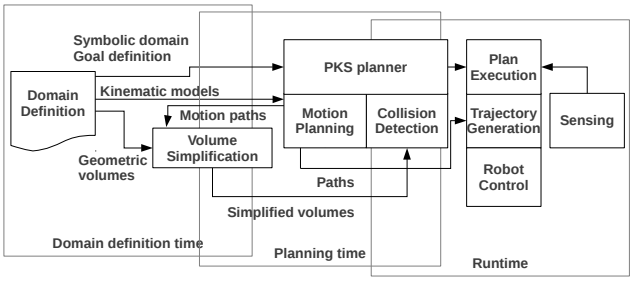


Figure 10: Overview of the implemented KVP framework (Gaschler et al. 2013a).

tended with several crucial additions to swept volume computations with sets of convex bodies (Gaschler et al. 2013a). To efficiently generate these sets of convex bodies, Mamou and Ghorbel's approximate convex decomposition algorithm (Mamou and Ghorbel 2009) is applied. An overview of KVP's component architecture is shown in Figure 10; the integration of PKS within this framework is achieved using the API described in this paper.

Finally, we note that the set of functions we defined for our planning API can be thought of as an interface to a series of abstract planning services which are ultimately implemented by some underlying "black box" planning system. As with other types of complex software modules, such an interface removes the need for the application programmer to know about how such services are actually implemented within the black box, but instead allows the designer to build more complex components that simply make use of these services. We are currently exploring the option of adapting other existing planners to use our interface, in order to experiment with alternative planner backends.

Conclusions

We described a set of extensions to PKS, aimed at improving its applicability to problems in robot task planning. We demonstrated the capabilities of our approach in solving typical robot tasks at the knowledge level, including the combination of high-level symbolic planning with low-level motion planning. Our evaluation included two simple scenarios that covered force sensing and visual sensing, with real execution on physical robot setups. A final example demonstrated a simple robot localisation task in simulation. As part of our ongoing and future work, we are continuing to refine our extensions and apply them in more complex scenarios, in order to gather empirical data and better understand the limits of our techniques. Overall, we believe our approach is useful for a broad range of robot planning applications that require incomplete knowledge, real-world geometry, and multiple robots and sensors.

Acknowledgements

This research was supported in part by the European Commission's Seventh Framework Programme under grant no. 270435 (JAMES, `james-project.eu`) and grant no. 270273 (XPERIENCE, `xperience.org`)

References

Barry, J. L. 2013. *Manipulation with Diverse Actions*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* 28(1):104–126.

Cheng, C.; Geisinger, M.; Ruess, H.; Buckl, C.; and Knoll, A. 2012. Game solving for industrial automation and control. In *IEEE Int. Conf. on Robotics and Automation*, 4367–4372.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009a. Semantic attachments for domain-independent planning systems. In *Proc. of the Int. Conference on Automated Planning and Scheduling (ICAPS)*, 114–121.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009b. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security & Rescue Robotics*, 1–6.

Edelkamp, S. 2002. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research* 20:195–238.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *The Semantic Web: Research and Applications*, 273–287.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Int. Conference on Robotics and Automation*, 4575–4581.

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, 115–125.

Etzioni, O.; Golden, K.; and Weld, D. 1994. Tractable closed world reasoning with updates. In *Proc. of the International Conference on Knowledge Representation and Reasoning (KR)*, 178–189.

Fikes, R., and Nilsson, N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Foster, M. E.; Gaschler, A.; Giuliani, M.; Isard, A.; Pateraki, M.; and Petrick, R. 2012. Two people walk into a bar: Dynamic multi-party social interaction with a robot agent. In *Proceedings of the ACM International Conference on Multimodal Interaction (ICMI)*.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Journal of Constraints, Special Issue on Constraints and Planning* 8:339–364.

Funge, J. 1998. Interval-valued epistemic fluents. In *AAAI Fall Symposium on Cognitive Robotics*, 23–25.

Gaschler, A.; Petrick, R.; Giuliani, M.; Rickert, M.; and Knoll, A. 2013a. KVP: A Knowledge of Volumes Approach to Robot Task Planning. In *IEEE/RSJ Intl Conf on Intelligent Robots and Systems (IROS)*, 202–208.

Gaschler, A.; Petrick, R.; Kröger, T.; Khatib, O.; and Knoll, A. 2013b. Robot task and motion planning with sets of convex polyhedra. In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.

Gaschler, A.; Petrick, R.; Kröger, T.; Knoll, A.; and Khatib, O. 2013c. Robot task planning with contingencies for run-time sensing. In *ICRA Workshop on Combining Task and Motion Planning*.

Giuliani, M.; Petrick, R.; Foster, M. E.; Gaschler, A.; Isard, A.; Pateraki, M.; and Sigalas, M. 2013. Comparing task-based and

socially intelligent behaviour in a robot bartender. In *Proceedings of the International Conference on Multimodal Interaction (ICMI)*.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1470–1477.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *International Journal of Robotics Research* 32(9–10):1194–1227.

Kress-Gazit, H., and Pappas, G. 2008. Automatically synthesizing a planning and control subsystem for the darpa urban challenge. In *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, 766–771.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence* 143:151–188.

Lozano-Pérez, T.; Jones, J.; Mazer, E.; and O'Donnell, P. 1989. Task-level planning of pick-and-place robot motions. *Computer* 22(3):21–29.

Mamou, K., and Ghorbel, F. 2009. A simple and efficient approach for 3d mesh approximate convex decomposition. In *IEEE International Conference on Image Processing (ICIP)*, 3501–3504.

Newell, A. 1982. The knowledge level. *Artificial Intelligence* 18(1):87–127.

Nilsson, N. 1984. Shakey the robot. Technical Report 323, AI Center, SRI International.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 324–332.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 212–221.

Petrick, R., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2–11.

Petrick, R.; Kraft, D.; Krüger, N.; and Steedman, M. 2009. Combining cognitive vision, knowledge-level planning with sensing, and execution monitoring for effective robot control. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, 58–65.

Petrick, R. 2011. An extension of knowledge-level planning to interval-valued functions. In *AAAI 2011 Workshop on Generalized Planning*.

Plaku, E., and Hager, G. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *IEEE Int. Conference on Robotics and Automation*, 5002–5008.

Poggioni, V.; Milani, A.; and Baioletti, M. 2003. Managing interval resources in automated planning. *Journal of Information Theories and Applications* 10:211–218.

Rickert, M. 2011. *Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems*. Dissertation, Technische Universität München.

van Eijck, J. 2013. Elements of epistemic crypto logic. Slides from a talk at the LogiCIC Workshop, Amsterdam.

Zacharias, F.; Borst, C.; and Hirzinger, G. 2006. Bridging the gap between task planning and path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4490–4495.

A Fast and Effective Online Algorithm for the Canadian Traveler Problem

O. Furkan Sahin and **Vural Aksakalli**
Dept. of Industrial Engineering
Istanbul Sehir University
34662 Istanbul, Turkey
furkansahin@std.sehir.edu.tr, aksakalli@sehir.edu.tr

Ali Fuat Alkaya
Dept. of Computer Engineering
Marmara University
34722 Istanbul, Turkey
falkaya@marmara.edu.tr

Abstract

The Canadian Traveler Problem (CTP) is a difficult path planning problem on stochastic graphs where some edges are blocked with certain probabilities and status of edges can be disambiguated only upon reaching an end vertex. The goal is to devise a policy that minimizes the expected traversal length between two given vertices. In this study, we introduce a simple, yet fast and effective sub-optimal algorithm for CTP that can be used in an online fashion. We present computational experiments involving real-world and synthetic data that suggest our algorithm finds near-optimal policies in very short execution times.

Introduction

The Canadian Traveler Problem (CTP) is a probabilistic path planning problem introduced by Papadimitrou and Yannakakis (1991). In this problem, an agent needs to travel from a source vertex s to a termination vertex t in a stochastic graph where some edges are blocked with certain probabilities and status of edges can be disambiguated only upon reaching an end vertex. The task here is to devise a strategy that will result in the shortest expected traversal length.

CTP has rather interesting characteristics in the sense that it can be cast both as a Markov Decision Process (MDP) with exponentially many states, or as a Partially Observable MDP (POMDP) with deterministic observations. Specifically, it can be shown that CTP belongs to an intermediate class of problems, called Deterministic POMDPs, which allow for state uncertainty but avoid noisy observations (Aksakalli and Sahin, 2014). In particular, CTP has been proven to be PSPACE-Complete (Fried et al., 2013), suggesting that not only its computational complexity is intractable, but its space complexity is intractable as well.

Despite its complexity, CTP finds practical applications in many different areas such as robot navigation (Blei and Kaelbling, 1999; Likhachev and Stentz, 2009), adaptive transportation systems (Fiosins et al., 2011), and minefield navigation (Fishkind et al., 2007). Amongst several variants of CTP, such as CTP with remote sensing (Bnaya, Felner, and Shimony, 2009), multi-agent CTP (Bnaya et al., 2011), CTP on disjoint-path graphs (Nikolova and Karger, 2008), our study focuses on the Discretized Stochastic Obstacle Scene Problem (D-SOSP), which is a variant on grid graphs with dependent edge probabilities (Aksakalli et al., 2011). D-SOSP is in fact a grid discretization of Continuous SOSP wherein an agent wishes to travel from one point to another in an obstacle field through arbitrarily-shaped regions, which may or may not be obstacles as specified through a certain probability function. The agent can disambiguate a possibly-obstacle region only upon reaching the region's boundary. The objective here is to find a policy that minimizes the expected length of traversal while deciding which regions to disambiguate and when. The reason we study D-SOSP variant of CTP is that we believe D-SOSP is perhaps the most realistic variant that has real-world applications in naval minefield navigation, a problem setting we consider in our computational experiments. We note that the difference between D-SOSP and general CTP is that in general CTP, there is no probabilistic dependency among edges. In other words, disambiguating one edge does not affect the status of other edges in CTP. However, in D-SOSP, edges intersecting the same obstacle are probabilistically dependent in the sense that (i) actual status of each edge intersecting the same obstacle are the same, and (ii) disambiguating any one of these edges will reveal the status of the other dependent edges as well.

There are several approximation and sub-optimal algorithms available in the literature for CTP (Baglietto et al., 2003; Xu et al., 2009; Eyerich, Keller, and Helmert, 2009) and there exist optimal algorithms for several special cases (Nikolova and Karger, 2008; Bnaya et al., 2011). In particular, Aksakalli (2007) proposes an exact algorithm for D-SOSP, called the BAO* Algorithm, which is an improvement on the classical AO* Search that uses stronger pruning techniques, including utilization of upper bounds on path lengths (in addition to lower bounds as in AO*), and uses less computational resources compared to AO*. On the other hand, Aksakalli and Sahin (2014) extends BAO* to general CTP and makes two key improvements: (1) they use a caching mechanism to avoid re-expansion of previously visited states, and (2) they make use of dynamic lower and upper bounds at a node level for state-space pruning. This new algorithm is called CAO*, which stands for AO* with Caching. CAO* is not polynomial-time, but it can significantly shorten the run time needed to find an exact solution to moderately-sized instances of the problem. Aksakalli

and Sahin (2014) illustrates that CAO* runs several orders of magnitude faster than BAO*, AO*, and value iteration. We use CAO* in our computational experiments for the purpose of finding the optimal policy.

Regarding sub-optimal algorithms for CTP, of particular interest is the Distance-to-Termination (DT) Algorithm that has been originally proposed for D-SOSP by Aksakalli and Ari (2014). This algorithm involves successive calculation of deterministic shortest paths with respect to a specific edge weight function during the agent's traversal. The authors present computational experiments that compare performance of the DT Algorithm against optimal policies obtained by the BAO* Algorithm on relatively small D-SOSP instances. Apart from DT, there are other heuristics for CTP in the literature as well. (Eyerich, Keller, and Helmert, 2009) evaluates these sampling-based heuristics both theoretically and empirically. Due to their sampling-based nature, they are likely to perform slower in comparison to penalty-based algorithms. However, quality-wise, there are currently no studies comparing their performances, which would be an excellent direction for future research.

The contribution of this study is two-fold: (1) we show how the DT Algorithm can easily be adapted for general CTP and, (2) we provide computational experiments to empirically assess performance of the DT Algorithm on the D-SOSP variant where the optimal policies are obtained by the CAO* Algorithm. In particular, CAO* allows us to solve much larger problem instances to better benchmark DT Algorithm's performance. We present experiments involving both real-world and synthetic data. Our results indicate that the DT Algorithm finds near-optimal policies in very short execution times and, its superior performance and computational savings are maintained on large problem instances as well. In what follows, we first provide formal definitions of CTP, Continuous SOSP, and D-SOSP. Next, we present adaptation of the DT Algorithm for general CTP, which is followed by our computational experiments.

CTP, Continuous SOSP, and Discretized SOSP

CTP Formulation

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be an undirected graph. An agent wishes to travel from $s \in \mathsf{V}$ to $t \in \mathsf{V}$ using edges $e \in \mathsf{E}$ for which the following functions are defined:

- weight function $\ell : \mathsf{E} \to \mathbb{R}_{\geq 0}$
- disambiguation cost function $c : \mathsf{E} \to \mathbb{R}_{\geq 0}$
- blockage probability function $p : \mathsf{E} \to [0, 1]$.

We assume that there are two types of edges. First, edges in the subset $\mathsf{E}' \subseteq \mathsf{E}$ are called *stochastic edges* for which traversability status are unknown, but blockage probabilities are known a priori by the agent in the form of the function p. The agent can not traverse a stochastic edge unless it has been disambiguated at a cost $c(e)$ and found to be unblocked. Disambiguation is defined as revealing the status of a stochastic edge by reaching an end vertex. Once disambiguated, status of a stochastic edge does not change over the course of traversal. It is further assumed that blockage probabilities of stochastic edges are independent. The second subset $\mathsf{E} \setminus \mathsf{E}'$ is called the set of *deterministic edges*, which are known to be traversable without any disambiguation requirements. For convenience, blockage probabilities of deterministic edges are defined to be 0. The Canadian Traveler Problem (CTP) is defined as finding the optimal policy that will result in the shortest expected $s - t$ path length. Since unlimited disambiguations will require the agent to visit all the vertices of the graph in the worst case, without loss of generality, we shall assume that there is a limit K on the total number of disambiguations the agent can perform. Indeed, total disambiguation requires visiting all the vertices of the graph in the worst case, thus the existence of the limit K.

Continuous SOSP Formulation

The Stochastic Obstacle Scene Problem (SOSP) is a variant of CTP that is a continuous-space path planning problem (Papadimitriou and Yannakakis, 1991). This problem is defined as follows: consider a marked point process in a region R in $\mathbb{R}^2$ which will be called as the *obstacle field*. The process creates random detections $X_T, X_F \subseteq R$ (corresponding to true and false detections, respectively) and marks $\rho_T : X_T \to [0, 1]$ and $\rho_F : X_F \to [0, 1]$. When a realization of this process occurs, only $X := X_T \bigcup X_F$ and $\rho := \rho_T \bigcup \rho_F$ are known to the agent at the planning time. It is assumed that $\rho(x)$ is the probability of $x \in X_T$, for all $x \in X$. Associated with each detection x is a region D_x that is possibly an obstacle. Without loss of generality, we assume these regions are open disks centered at x with a fixed radius of $r > 0$. In SOSP context, for any $x \in X$, the probability $\rho(x)$ is referred to as "mark" of the associated disk D_x, which is defined as the probability of the given disk being a true obstacle.

At planning time, the agent knows the mark of each possible-obstacle, but not their actual true/ false status. Starting from a point $s \in R$, towards the termination $t \in R$, the agent wants to travel a continuous s, t curve in $(\bigcup_{x \in X_T} D_x)^C$ with the shortest possible arclength (in this context, C denotes the set complement operator). Furthermore, it is assumed that there is a dynamic learning capability which is defined as the option to *disambiguate* the obstacle (to reveal if $x \in X_T$ or not) when the curve is on the boundaries ∂D_x. However, given a cost function $c : X \to \mathbb{R}_{\geq 0}$, a cost $c(x)$ is added to the overall length of path. We assume that the agent has a disambiguation limit of K. The strategic decision of where and when to disambiguate in order to minimize the expected length of the curve is called the *Continuous SOSP*.

Discretized SOSP Formulation

Optimal disambiguation algorithms are capable of solving only the most trivial instances of continuous SOSP. Hence, for the simplicity and convenience, a discrete approximation of continuous SOSP is considered. In this approximation, we define a graph G consisting of vertices that correspond to all pairs of integers i, j such that $1 \leq i \leq i_{\text{max}}$ and $1 \leq j \leq j_{\text{max}}$, where i_{max} and j_{max} are integers specifying boundaries of the obstacle field. The lattice edges are of the following four types:

1. Between (i, j) and $(i + 1, j)$ with unit length,
2. between (i, j) and $(i, j + 1)$ with unit length,
3. between (i, j) and $(i + 1, j + 1)$ with length $\sqrt{2}$,
4. between $(i + 1, j)$ and $(i, j + 1)$ with length $\sqrt{2}$.

A vertex s is designated as the starting vertex and another vertex t as the termination. The agent seeks to traverse from s to t in G through "traversable edges", which are the ones that do not intersect with a true or ambiguous obstacle. It is possible for the agent to traverse the edges that intersect false obstacles. If, however, the agent seeks to use an edge that is intersecting with an ambiguous obstacle, a disambiguation should be carried on one of the outermost edges of the ambiguous obstacle. The goal here is to devise a policy that will minimize the expected length of traversal by effective exploitation of the disambiguation capability. We refer to this discretized problem as *Discretized SOSP*, or simply D-SOSP. It is important to observe that D-SOSP is simply a variant of CTP on grid graphs with probabilistic dependency among stochastic edges. A simple D-SOSP instance is shown in Figure 1.

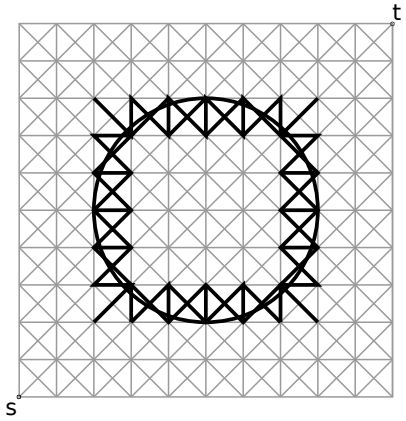


Figure 1: Lattice discretization of a simple SOSP instance with only one disk. Stochastic edges, i.e., edges intersecting with the disk, are shown in bold.

The DT Algorithm

First introduced by Aksakalli and Ari (2014), the notion of penalty-based algorithms for D-SOSP is a heuristic framework that involves successive calculation of deterministic shortest paths with respect to a specific edge weight function during the agent's $s - t$ traversal. The idea behind using an edge weight function is to discourage traversing stochastic edges by assigning them additional weights. A penalty-based algorithm within the context of CTP employs the navigate-disambiguate-repeat (NDR) strategy described below:

1. Find the deterministic shortest path from start s to termination t in the graph where all the edge weights are assigned by the weight function.
2. Traverse the path until a vertex associated with an ambiguous stochastic edge is reached.
3. Since an ambiguous edge cannot be traversed, disambiguate the edge from the current vertex. Set the blockage probability to zero if the edge has been found to be traversable, and 1 otherwise.
4. Set the current vertex as the new starting vertex s and repeat 1 through 3 above until t is reached.

Aksakalli and Ari (2014) generalizes the weight functions utilized in NDR strategy, using the notion of "penalty functions":

$$w_D^F(e) := \ell^{\mathsf{e}}(e) + \mathbf{1}_{e \in \mathsf{E}'} \cdot F(e), \tag{1}$$

As a result, it is now possible to plug in different penalty functions to obtain different weight calculations. In fact, apart from DT Algorithm, the article includes an extensive discussion of two other penalty functions, Simulated Risk Disambiguation Algorithm (SRA) (Fishkind et al., 2007) and Reset Disambiguation Algorithm (RDA) (Aksakalli et al., 2011). In SRA, the penalty function F is specified as $F_{SR}(e) := \alpha \log(1 - \rho(e))^{-1}$ whereas it is defined as $F_{RD}(e) := \frac{c(e)}{1-\rho(e)}$ for RDA. The first function is motivated by the idea of risk simulation (temporarily assuming that ambiguous edges are riskily traversable), where the second function is based on the idea of using the optimal weights for parallel graphs on arbitrary instances. A major disadvantage of SRA is that it requires to tune the parameter α for improved performance, thus, increased computational time. The lack of a tuning parameter, as it has been empirically shown, provides a significant advantage for RDA in terms of run time. However, despite its better performance and lack of tuning parameters, the weight function F_{RD} cannot be used when the disambiguation cost is zero. In other words, in a setting where the agent performs the disambiguation by simply a clear line of sight, F_{RD} is not applicable.

The above mentioned disadvantages of F_{SR} and F_{RD} reveals the quest to find a better penalty function. After extensive computational experiments, Aksakalli and Ari (2014) observed that the penalty function $F_{DT}(e) := c(e) + \left(\frac{d_t(e)}{1-\rho(e)}\right)^{-\log(1-\rho(e))}$ consistently outperformed both of the former functions in most of the instances. The new function utilized the cost parameter as an additive term and it was monotonically nondecreasing in $c(e)$ and $\rho(e)$ for edges that intersect possible-obstacles in discretized SOSP. In particular DT algorithm uses the following weight for D-SOSP:

$$w_D^{DTA}(e) := \ell(e) + \mathbf{1}_{e \in \mathsf{E}'} \cdot \Big(c(e) + \Big(\frac{d_t(e)}{1 - p(e)} \Big)^{-\log(1-p(e))} \Big)$$

Above, $\mathbf{1}$ is the indicator function and $d_t(e)$ denotes the distance of edge e's midpoint to t, hence the name "distance-to-termination". The DT Algorithm thus calculates at most K deterministic paths and therefore it is extremely fast. It can also be used in an online fashion as the agent traverses the graph. Note, however, that computation of the expected path length requires $O(2^K)$ path calculations. The authors,

however, make the following observation regrading the DT Algorithm:

"Despite the fact that DTA performed remarkably well [...] in our simulations, it may or may not perform at the same level on obstacle fields with different topologies or with non-circular obstacle regions. Further research on instances with different characteristics is required in order to confirm that high performance of DTA is consistent across various problem settings. To that end, it might as well be the case that perhaps a different penalty function outperforms that of DTA in certain problem environments. Nonetheless, the NDR strategy guided by appropriate penalty functions seems to be an efficient and effective algorithmic framework for SOSP, and our study could be seen as a show case of this framework using the DT penalty function on an important real-world variant of the problem".

CAO*

The CAO* algorithm is an improvement upon the classical AO* Algorithm (Chang and Slagle, 1971; Martelli and Montanari, 1978) for searching AO trees. By utilizing admissible lower bounds called heuristic labels that are guaranteed not to overestimate the true label of any node, the AO* Algorithm guides the search so that it is only required to examine a small portion of the complete AO tree. Although the labels are referred to as "heuristics" in the AO* terminology, the AO* algorithm itself is optimal with the prerequisite of having admissible lower bounds.

CAO* exploits the following two important properties of CTP: (1) admissible upper bounds, (2) state overlaps in the AO tree. First, by utilizing the fast and efficient DT Algorithm, CAO* can quickly determine an upper bound that is very close to the optimal, which results in pruning a large portion of the solution tree. Second, when disambiguation limit K is greater than 1, the agent might end up at the same particular state in the AO tree after visiting different sequences of states. To take advantage of this property, CAO* uses a state caching mechanism to avoid generation of multiple copies of same nodes. As a result, CAO* examines a very small fraction of the complete tree, thus decreasing the run time required to obtain a solution. In fact, computational experiments showed that CAO* executed 770 times faster than value iteration and 1,850 times faster than the classical AO* algorithm on D-SOSP problems. For further discussion, the reader is referred to (Aksakalli and Sahin, 2014).

On a related note, Kuter and Hu (2008) describe a general methodology for reducing the search space in factored MDPs via upper and lower bounds on the value function using state equivalence classes. CAO* Algorithm also benefits from such bounds, yet these bounds take advantage of the special problem structure in CTP and therefore they are much tighter than those suggested in Kuter and Hu (2008).

Computational Experiments

In this section, performance of the DT Algorithm is empirically compared to CAO* on the D-SOSP variant of CTP. The computational experiments are conducted in a maritime minefield navigation domain. Our simulations were performed in two different environments: Environment 1 that is concerned with a real-world data set, and Environment 2 that involves synthetic data. In both environments, we consider cases with disambiguation limit $K = 1, \ldots, 5$ and disambiguation cost $c = 0, 2, 4, 6$.

Environment 1

In the first environment, we consider a U.S. Navy minefield data set, called COBRA data, which was used in Priebe et al. (2005); Fishkind et al. (2007); Aksakalli et al. (2011); Aksakalli and Ceyhan (2012). This data set has 39 disk-shaped potential obstacles with disk radius $r = 5$ on a 100×100 integer lattice. A visual representation of the COBRA environment is shown in Figure 2.

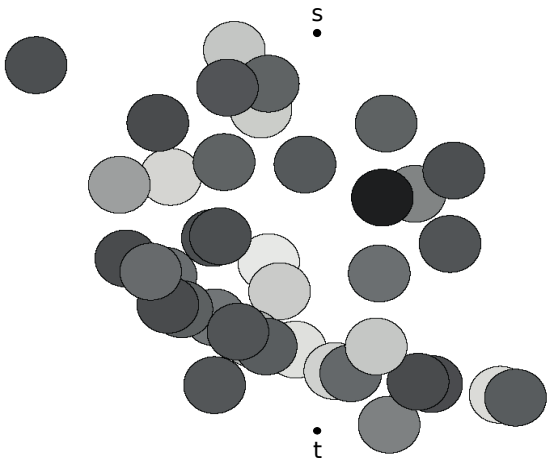


Figure 2: Illustration of COBRA data set. The gray intensity of disks reflect probability of the disks being true obstacles.

Table 1 shows the experiment results performed on the COBRA data set where the "zero-risk" column denotes the length of the $s-t$ path avoiding all disks without performing any disambiguations. On the average, policies found by the DT Algorithm was only 1.3% worse than the optimal policy, yet mean DT run time was 7.8 seconds, which is about 200 times faster than CAO*. In fact, median percent difference for DT in terms of expected path length was merely 0.3%.

Environment 2

In Environment 2, we randomly sampled six "COBRA-like" instances with 39 disks with a radius of 5 units on a 100×100 integer lattice. To make the environment even more challenging, the instances were conditioned to have a zero-risk path length of at least 130 units. The results are shown in Table 2. Similar to Environment 1, DT Algorithm found solutions very close to the optimal within very short execution times. On the average, policies found by the DT Algorithm was 3.17% worse than the optimal policy. However, mean DT run time was 8.15 seconds, which is about 740 times faster than CAO*. In fact, DT Algorithm ran up to 3300 times faster than CAO*. On the other hand, median percent difference for DT in terms of expected path length

Table 1: Performance of the DT Algorithm on COBRA data set for the K, c combinations listed.

		Expected Distance (units)				Run Time (seconds)		
K	c	Zero-Risk	OPT	DTA	Percent Difference	OPT	DTA	Ratio
1	0	104.33	80.02	80.17	0.18	7.32	3.46	2.12
	2	104.33	82.02	82.17	0.18	5.47	3.21	1.70
	4	104.33	84.02	84.17	0.17	8.39	3.71	2.26
	6	104.33	86.02	86.17	0.17	8.04	3.68	2.18
2	0	104.33	75.47	80.25	6.34	389.80	4.50	86.62
	2	104.33	79.47	79.74	0.34	1404.12	4.72	297.48
	4	104.33	81.77	81.94	0.21	707.33	4.47	158.24
	6	104.33	83.98	84.15	0.21	1422.32	4.81	295.70
3	0	104.33	74.20	78.20	5.39	935.58	9.49	98.59
	2	104.33	79.27	79.78	0.63	4261.26	10.43	408.56
	4	104.33	81.73	82.02	0.36	1582.92	9.70	163.19
	6	104.33	83.97	84.27	0.35	1304.64	9.56	136.47
4	0	104.33	73.81	76.93	4.23	1736.94	9.80	177.24
	2	104.33	79.02	79.54	0.66	4241.33	11.08	382.79
	4	104.33	81.56	81.82	0.31	2579.95	10.01	257.74
	6	104.33	83.85	84.09	0.29	2224.39	10.22	217.65
5	0	104.33	73.51	76.93	4.64	2291.41	10.10	226.87
	2	104.33	79.01	79.54	0.67	4992.36	11.92	418.82
	4	104.33	81.56	81.82	0.32	3802.77	11.21	339.23
	6	104.33	83.85	84.09	0.29	3202.90	11.08	289.07
Mean		104.33	80.40	81.39	1.30	1855.46	7.86	198.13
Std.		0.00	3.72	2.59	2.02	1565.04	3.25	138.14
Median		104.33	81.56	81.82	0.32	1502.62	9.63	156.03

was only 0.96%. It can be also observed that computational benefits of DT Algorithm get more significant as the disambiguation limit K is increased.

Conclusions

CTP is a difficult stochastic path planning problem and D-SOSP is perhaps the most realistic variant of CTP. These problems have practical applications in robot navigation, adaptive traffic routing, and mine-field navigation. In this study, we consider the DT Algorithm for CTP, which is a sub-optimal online algorithm that is fast and effective. This algorithm involves successive calculation of deterministic shortest paths with respect to a certain edge weight function during the agent's traversal. We provide computational experiments to empirically assess performance of the DT Algorithm on the D-SOSP variant. In our experiments, the optimal policies are obtained by the CAO* Algorithm, which is a state-of-the-art exact algorithm for CTP based on the classical AO* Search. We present computational experiments involving both real-world and synthetic data. Our results indicate that the DT Algorithm finds near-optimal policies in very short execution times. Computational benefits of the DT Algorithm become even more significant as the problem instances get larger. In particular, our results show that percent deviation from the optimal policies found by the DT Algorithm can be as low as 0.2%, and DT can run up to 3300 times faster than CAO*.

Acknowledgements

CTP, Continuous SOSP, and D-SOSP formulations are adapted from Aksakalli and Ari (2014). Work of V. Aksakalli and O.F. Sahin was supported by The Scientific and Technological Research Council of Turkey (TUBITAK), Grant 111M541. Work of A.F. Alkaya was supported by Marmara University Scientific Research Committee.

References

Aksakalli, V., and Ari, I. 2014. Penalty-based algorithms for the stochastic obstacle problem. *In Press, INFORMS J. on Computing, DOI:10.1287/ijoc.2013.0571*.

Aksakalli, V., and Ceyhan, E. 2012. Optimal obstacle placement with disambiguations. *Ann. Appl. Stat.* 6(4):1730–1774.

Aksakalli, V., and Sahin, O. 2014. An AO* based exact algorithm for the Canadian traveler problem. *Submitted for Publication*.

Aksakalli, V.; Fishkind, D.; Priebe, C.; and Ye, X. 2011. The reset disambiguation policy for navigating stochastic obstacle fields. *Naval Res. Logist.* 58:389–399.

Aksakalli, V. 2007. The BAO* algorithm for stochastic shortest path problems with dynamic learning. In *In Proc. the 46th IEEE Conf. on Decision and Control, New Orleans, LA*, 6003–6008. Hoboken, NJ: Wiley-IEEE Press.

Baglietto, M.; Battistelli, G.; Vitali, F.; and Zoppoli, R. 2003. Shortest path problems on stochastic graphs: a neuro dynamic programming approach. In *In Proc. the 42nd IEEE Conf. on Decision and Control*, 6187–6193. Hoboken, NJ: Wiley-IEEE Press.

Blei, D., and Kaelbling, L. 1999. Shortest paths in a dynamic uncertain domain. In *In Proc. IJCAI Workshop*

Table 2: Average performance of the DT Algorithm on six COBRA-like data sets for the K, c combinations listed.

		Expected Distance (units)				Run Time (seconds)		
K	c	Zero-Risk	OPT	DTA	Percent Difference	OPT	DTA	Ratio
1	0	138.27	119.21	132.70	11.32	10.01	3.44	2.91
	2	138.27	121.21	134.70	11.13	10.44	4.01	2.60
	4	138.27	123.21	137.03	11.22	10.12	4.03	2.51
	6	138.27	125.21	131.01	4.63	9.94	3.76	2.64
2	0	138.27	110.52	112.16	1.49	1214.73	4.22	287.85
	2	138.27	113.58	114.96	1.21	1376.05	5.13	268.24
	4	138.27	116.38	116.83	0.39	1413.53	5.13	275.54
	6	138.27	119.17	123.63	3.75	1609.05	5.59	287.84
3	0	138.27	107.72	109.71	1.85	6179.67	11.02	560.77
	2	138.27	111.21	112.65	1.29	5249.41	9.98	525.99
	4	138.27	114.36	115.50	1.00	4987.30	9.72	513.10
	6	138.27	117.34	118.68	1.15	4808.78	9.79	491.19
4	0	138.27	106.22	109.10	2.71	17609.29	11.45	1537.93
	2	138.27	110.76	112.10	1.20	12125.10	11.37	1066.41
	4	138.27	113.97	115.01	0.91	8897.27	10.68	833.08
	6	138.27	116.97	118.04	0.91	7911.87	10.04	788.03
5	0	138.27	105.54	109.00	3.27	35590.07	10.76	3307.63
	2	138.27	110.17	112.03	1.69	17621.04	10.92	1613.65
	4	138.27	113.45	114.69	1.09	13920.19	10.92	1274.74
	6	138.27	116.53	117.80	1.09	13065.81	10.94	1194.31
Mean		138.27	114.64	118.37	3.17	7680.98	8.15	741.85
Std.		0.00	5.42	8.75	3.63	8832.52	3.19	788.59
Median		138.27	114.16	115.26	0.96	5118.35	9.89	517.79

on Adaptive Spatial Representations of Dynamic Environments. Palo Alto, CA: AAAI Press.

Bnaya, Z.; Felner, A.; Fried, D.; Maksin, O.; and Shimony, S. 2011. Repeated-task Canadian traveler problem. In *In Proc. 4th Annual Symposium on Combinatorial Search*, 24–30. Palo Alto, CA: AAAI Press.

Bnaya, Z.; Felner, A.; and Shimony, S. 2009. Canadian traveler problem with remote sensing. In *In Proc. IJCAI 2009*, 437–442. AAAI Press.

Chang, C., and Slagle, J. 1971. An admissible and optimal algorithm for searching and/or graphs. *Artificial Intelligence* 2:117–128.

Eyerich, P.; Keller, T.; and Helmert, M. 2009. High-quality policies for the Canadian traveler problem. In *In Proc. the 24th AAAI Conf. on Artificial Intelligence, Atlanta, Georgia*, 51–58. Palo Alto, CA: AAAI Press.

Fiosins, M.; Fiosina, J.; Müller, J.; and Görmer, J. 2011. Reconciling strategic and tactical decision making in agent-oriented simulation of vehicles in urban traffic. *In Proc. the 4th Internat. ICST Conf. on Simulation Tools and Techniques* 144–151.

Fishkind, D.; Priebe, C.; Giles, K.; Smith, L.; and Aksakalli, V. 2007. Disambiguation protocols based on risk simulation. *IEEE Trans. on Systems, Man, and Cybernetics, Part A* 37(5):814–823.

Fried, D.; Shimony, S.; Bensaat, A.; and Wenner, C. 2013. Complexity of Canadian traveler problem variants. *Theoretical Comp. Sci.* 487:1–16.

Kuter, U., and Hu, J. 2008. Computing and using lower and upper bounds for action elimination in MDP planning. In *In Proc. the 7th Symposium on Abstraction, Reformulation, and Approximation (SARA-07), Whistler, Canada*.

Likhachev, M., and Stentz, A. 2009. Probabilistic planning with clear preferences on missing information. *Artificial Intelligence* 173:696–721.

Martelli, A., and Montanari, U. 1978. Optimizing decision trees through heuristically guided search. *Comm. ACM* 21:1025–10039.

Nikolova, E., and Karger, D. R. 2008. Route planning under uncertainty: the Canadian traveller problem. In *In Proc. the 23rd AAAI Conf. on Artificial Intelligence, Chicago, Illinois*, 969–974. Palo Alto, CA: AAAI Press.

Papadimitriou, C., and Yannakakis, M. 1991. Shortest paths without a map. *Theoretical Comp. Sci.* 84:127–150.

Priebe, C.; Fishkind, D.; Abrams, L.; and Piatko, C. 2005. Random disambiguation paths for traversing a mapped hazard field. *Naval Res. Logist.* 52:285–292.

Xu, Y.; Hu, M.; Su, B.; Zhu, B.; and Zhu, Z. 2009. The Canadian traveller problem and its competitive analysis. *J. Combinatorial Opt.* 18:195–205.

Collision-free Path Planning for Remote Laser Welding

András Kovács
Fraunhofer Project Center for Production Management and Informatics,
Computer and Automation Research Institute, Budapest, Hungary
andras.kovacs@sztaki.mta.hu

Abstract

The paper proposes algorithms for collision-free path planning in robotic Remote Laser Welding (RLW), using collision detection on a triangle mesh representation of the moving objects and a path planning algorithm based on a classical A* search, both highly specialized to the needs of RLW. The algorithms depart from an optimized task sequence and an initial, potentially colliding rough-cut path. The algorithms modify this path to eliminate all collisions while preserving the stitch sequence and minimizing the cycle time. The approach is validated in computational experiments on real industrial data involving the welding of a car front door.

Introduction

A recent technological trend in the assembly of sheet metal parts, such as car bodies, is the spreading application of Remote Laser Welding (RLW). This contactless technology eliminates the most important limitation of earlier joining techniques, the accessibility issues between the welding gun and the workpiece, by welding from a distant point using a laser beam emitted from a laser scanner that is moved by an industrial robot. This results in up to 80% lower cycle times, reduced operating costs, and higher freedom in part design (Park and Choi 2010). For conventional machining technologies, on-line programming by manual guidance of the robot is the typical programming approach. However, due to the redundancy in the degrees of freedom of the RLW robot and the laser scanner, on-line programming is hardly possible for RLW. However, efficient off-line programming methods tailored to the needs of RLW hardly exist (Reinhart, Munzert, and Vogl 2008).

Our general objective is the development of an interactive off-line programming toolbox with efficient optimization capabilities for RLW (Erdős et al. 2013). In a recent paper (Kovács 2013), we have introduced an efficient algorithm for integrated task sequencing and rough-cut path planning. Significant novelties of the algorithm include (1) the explicit modeling of the coupled movement of the quick tool (laser beam repositioning) and the relatively slow robot; (2) exploiting the high degree of freedom in choosing the robot path when welding a well-defined stitch position; and finally, (3) planning in the continuous space, without losses stemming from sampling that characterizes many other approaches working on a discretized space representation. Nevertheless, that algorithm ignores potential collisions along the path. This assumption is not absolutely unrealistic, since general (fixture) design guidelines for RLW require that the access volumes of the welding stitches are left clear to preclude collisions. However, our experience on real industrial data showed that this requirement is sometimes overridden by other design objectives, and collisions do occur on the computed rough-cut path.

In this paper, we propose a collision-free path planning algorithm that departs from the above rough-cut path, and modifies it to avoid any collisions while preserving the original stitch sequence and minimizing the cycle time. We present collision detection techniques on triangle mesh models and a path planning algorithm based on classical A* search, both highly specialized to the needs of RLW. These include searching for a trajectory that visits given regions of the 3D space in a pre-defined order and spends the time required to execute the corresponding actions in each of those regions. The definition of collision depends on the action executed in a given position. A trajectory that minimizes the cycle time is looked for.

The paper is organized as follows. First, a brief review of the related literature and the technological background is given. Then, the path planning problem is defined formally. Afterwards, the proposed collision detection and path planning methods are presented in detail. Finally, computational experiments are reported and conclusions are drawn.

Literature Review

The RLW technology, including its benefits and limitations, is presented in (Tsoukantas et al. 2007). Applications of RLW in the automotive industry are reviewed in (Shibata 2008). The importance of automated process planning for RLW is emphasized in (Hatwig, Reinhart, and Zaeh 2010).

We are aware of a single earlier approach to task sequencing and path planning specifically for RLW and remote laser cutting, introduced in a series of papers (Reinhart, Munzert, and Vogl 2008; Hatwig et al. 2012). The proposed algorithms are designed mostly for planar workpieces: task sequencing is performed by solving a traveling salesman problem (TSP) over the fixed welding stitch positions, and a robot path is computed in a plane above the workpiece. Potential collisions are ignored. A similar model is applied

and construction heuristics are proposed for path planning in laser cutting in (Dewil, Vansteenwegen, and Cattrysse 2014). The applied model also captures sophisticated ordering constraints among the contours to be cut.

An efficient, generic task sequencing and collision-free path planning model, with illustrations from resistance spot welding (RSW) is presented in (Saha et al. 2006). A critical assumption is that the robot can execute each effective task from a relatively small set of candidate configurations, e.g., at most 10 configurations per task, which can be generated a priori. An iterative algorithm is proposed that tries to compute as few point-to-point collision-free paths as possible, hence avoids solving unnecessary computationally demanding subproblems. The difficulty in applying this approach to RLW stems from the fact that efficient paths in RLW exploit the free movement of the robot in the continuous space while welding.

The minimization of processing time in a milling operation is investigated in (Castelino, D'Souza, and Wright 2002). A Generalized TSP (GTSP) approach is proposed, where the nodes correspond to the candidate tool entry/exit points for machining a feature. Potential collisions are neglected. A TSP with Neighborhoods (TSPN) model is proposed in (Alatartsev et al. 2013) for sequencing a set of robotic tasks whose start/end points can be chosen arbitrarily along open or closed contours, such as in the case of different cutting problems. In (Alatartsev, Augustine, and Ortmeier 2013) a construction heuristic, the so-called constricting insertion heuristic is introduced for the derived TSPN over a set of 2D polygons. A multi-objective constraint optimization model is proposed in (Kolakowska, Smith, and Kristiansen 2014) for task sequencing in spray painting, for minimizing cycle time and maximizing paint quality at the same time.

Classical AI methods for collision-free path planning search a discretized grid representation of the environment using algorithms like A* or one of its numerous descendants. These include D*-Lite (Koenig and Likhachev 2005) or Focussed D* (Stentz 1995) for dynamically changing environments, or ARA* (Likhachev, Gordon, and Thrun 2003), an anytime algorithm with provable bounds on sub-optimality. Field D* (Ferguson and Stentz 2006) lifts the constraint on the previous algorithms that they must move through a series of neighboring grid points, thus saving unnecessary turnings and further reducing path length. These methods are suitable mostly for lower dimensional problems due to the computational effort required.

Higher dimensional problems, such as path planning for a robot with many degrees of freedom, are often intractable using the above methods. In such cases, incomplete methods that apply randomization are preferred. The most efficient approaches are *Rapidly-exploring Random Trees* (RRT) (Kuffner and LaValle 2000) for the single-query case, and *Probabilistic Roadmaps* (PRM) (Kavraki et al. 1996; Geraerts and Overmars 2002) for the multiple-query case. A recent tendency is to delegate motion planning to GPUs, see, e.g., (Park et al. 2013), where a highly parallelized RRT algorithm is proposed to exploit the computation capabilities of GPUs.

Path planning algorithms typically rely on external software libraries for collision detection. Such libraries contain, e.g., the Proximity Query Package (PQP) (Larsen et al. 2000) for rigid objects represented as triangle mesh, or V-Collide (Hudson et al. 1997) specifically for VRML applications. A benchmarking suite for pairwise static collision detection algorithms and a comparison of numerous freely available collision detection algorithms has been presented in (Trenkel, Weller, and Zachmann 2007).

The integration of task and motion planning has received significant attention in the robotics community, especially in navigation and manipulation applications. A plethora of approaches has been offered to combine symbolic planners as high-level solvers and motion planners (e.g., PRM or RRT planners) as subproblem solvers, see, e.g., (Kaelbling and Lozano-Perez 2011; Srivastava et al. 2014).

Technological Background

The Welding Process

The recent development of a new generation of laser sources, such as fiber lasers, enabled laser welding with an operating distance (focal length) above one meter. The new technology, RLW, joins sheet metal parts without physical contact or even a close approach. This, on the one hand, ensures extremely fast positioning speed compared to classical RSW, where a vast welding gun must contact the workpiece. The high productivity of the technology results in up to 80% lower cycle times and reduced operating costs, making RLW economically profitable despite the high initial investments. In addition to the direct economic gain, the abolishment of the accessibility issues removes many earlier constraints on part designs, an advantage that can be turned easily into parts with reduced weight, yet higher stiffness. This, in the automotive industry, facilitates the design of lighter and more efficient cars, without compromising safety.

An RLW operation consists in joining two or more sheet metal parts at various joints. In this paper, we assume stitch welds, i.e., linear welding stitches with a typical length of 15-30 mm each. During the operation, the parts are held in a fixture. It is assumed that the operation is performed by a single RLW robot. A typical RLW robot consists of a robot arm with 4 rotational joints and a laser scanner. The robot arm moves the scanner with a maximum speed of 0.2-0.6 m/s, and due to the low scanner weight, with a rather high acceleration. The scanner contains two tilting mirrors for the rapid positioning of the laser beam (up to 5 m/s), and a lens system to regulate the focal length. Hence, the typical RLW robot is a redundant kinematic chain with 7 degrees of freedom, in which the mirrors in the scanner position the laser beam an order of magnitude faster than the movement of the mechanical joints of the robot arm. The process of welding a car door by an RLW robot is depicted in Figure 1.

The robot can weld a stitch if the scanner is located within the *focus range* (e.g., 800-1200 mm) from the stitch, and the *inclination angle* (i.e., the angle between the laser beam and the surface normal) is not more than a specified technological parameter (e.g., 15°). These constraints define a truncated cone above the stitch, which will be called the *tech-*

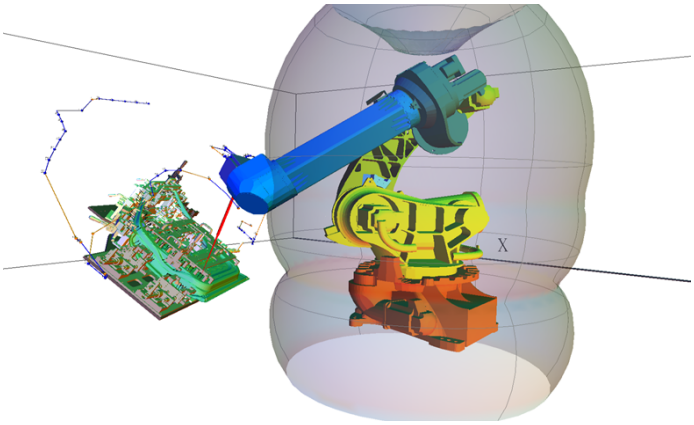


Figure 1: RLW robot welding a car front door, positioned in a fixture. The blue sections of the indicated scanner path represent the movement of the robot while welding, while yellow sections denote idle movement.

nological access volume (TAV) of the stitch, as shown in Figure 2. Strictly speaking, the above definition would require spherical outer and inner bases for the truncated cone. However, to benefit from convex TAVs, we approximate this shape by using a planar inner base, while leaving a spherical outer base. The *collision-free access volume* is the subset of the TAV from which welding can be performed without collisions. Since the length of a stitch is significantly smaller than other characteristic dimensions in the welding process, it is reasonable to assume that all points of a stitch can be processed from the access volume belonging to the mid-point of the stitch.

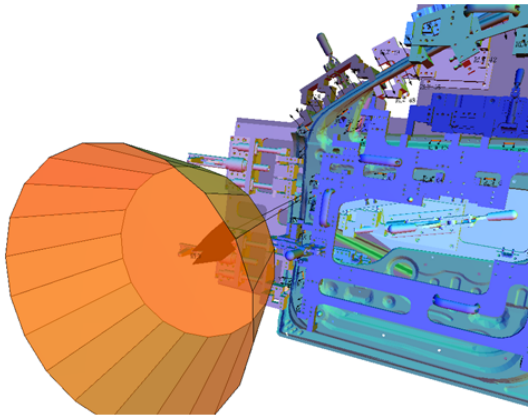


Figure 2: Technological access volume (TAV) of a welding stitch.

Each stitch can be welded at a given speed (e.g., 50 mm/s), which depends on the thickness and the material of the parts to join. Each stitch must be processed without interruption. The robot can weld the stitch while in motion, therefore the trajectory of the scanner must be a curve in the 3D space, such that sufficient time is spent in the access volume of each stitch. There are 30-75 stitches to weld in an RLW operation in the automotive industry.

An Off-line Robot Programming Approach

In industrial practice, robot programming is still typically performed by on-line programming, i.e., by manually guiding the robot from one position to the next, at very small steps, which is a extremely time consuming and hardly feasible for RLW. Our goal is to implement a complete off-line programming toolbox for RLW, which can provide an automated method for computing close-to-optimal robot programs. This involves the optimization of the task sequence, integrated with rough-cut path planning; collision-free path planning in the workpiece coordinate system; the placement of the workpiece in the welding cell; the inverse kinematic transformation that converts the path into the robot joint coordinate system; and finally, the simulation of complete process plan and the automated generation of the robot program code (see Figure 3). The workflow has been presented in detail in (Erdős et al. 2013).

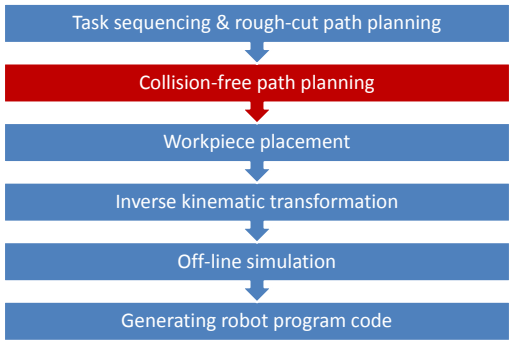


Figure 3: Workflow in the off-line programming system. The paper focuses on the second step, collision-free path planning.

An important consequence of the above workflow is that path planning is performed in the 3D Cartesian coordinate system of the workpiece. This was motivated by the fact that the extensive geometric computations required for the optimization of the task sequence and the robot path cannot be executed efficiently in the robot joint coordinate system (Kucuk and Bingul 2006).

In the recent conference paper (Kovács 2013) we have presented an efficient algorithm for integrated task sequencing and rough-cut path planning, using a detailed technological model of the RLW process. However, that algorithm ignores potential collisions along the path, exploiting that RLW is less exposed to accessibility issues than any other welding technology, and hence, the optimal task sequence is hardly affected by collisions. The objective of the path planning algorithm investigated in this paper is eliminating all collisions from the rough-cut path while preserving the given task sequence and minimizing cycle time.

Problem Definition

The collision-free path planning problem consists in computing a scanner trajectory in the 3D Cartesian coordinate

system attached to the workpiece, such that the robot welds all stitches along the path and the cycle time is minimized. Formally, there is a list of n welding stitches, denoted by $(s_1, s_2, ..., s_n)$, to be welded by an RLW robot in this predefined order, originally computed by some task sequencing algorithm. Each stitch is characterized by its technological access volume, TAV_i, a truncated cone as defined above, a collision-free access volume, $\text{CFAV}_i \in \text{TAV}_i$, and the associated welding time, t_i. Each stitch s_i must be welded without interruption, during which the movement of the scanner is constrained to CFAV_i. Only one stitch can be welded at a time. The path may contain idle robot movement, i.e., sections without welding. Such sections of the path must be located within CF_0, the region that is free of collisions of the robot (with the laser beam switched off).

It is assumed that the maximum robot speed (speed of the scanner), v, is independent of the position in the working area, and the robot has an infinite working area. Finally, the objective is minimizing the cycle time, i.e., the total time required for the robot to travel along the computed trajectory.

Path planning must avoid all types of collisions that can be detected at this phase of the workflow, i.e., that are independent of decisions made in later phases (see Figure 3 earlier). These are the collisions between the *laser beam* vs. the *workpiece* and the *fixture*, as well as the *scanner head* vs. the *workpiece* and the *fixture*. It is noted that these are the most critical types collisions in RLW.

In addition, we assume that there is given an initial, potentially colliding rough-cut path, which has been originally computed by an external algorithm, practically, the earlier proposed task sequencing and rough-cut path planning algorithm. This initial trajectory welds each stitch from TAV_i, but potentially from outside CFAV_i. Below, we propose a procedure that detects collisions along the rough-cut path, and resolves those collisions by a series of modifications to the initial path. The result of applying this method for collision avoidance is shown in Figure 4.

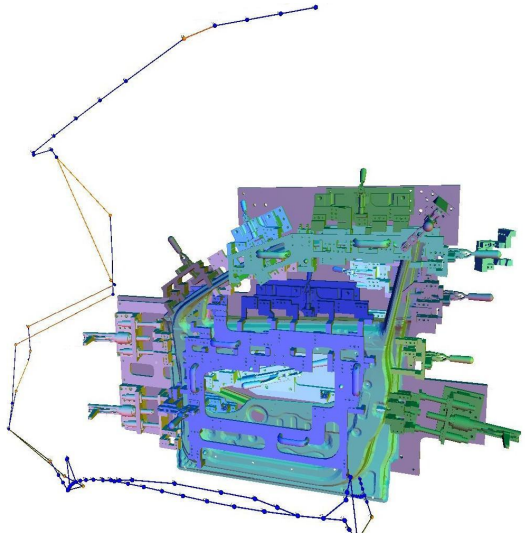


Figure 4: Comparison of the rough-cut and the collision-free paths. Blue sections denote welding, while yellow section correspond to idle movement.

Collision detection

Collision detection is performed using PQP (Larsen et al. 2000) on a triangle mesh representation of the involved 3D objects. The mesh representation of the workpiece and the fixture is given as input, in STL file format, whereas the mesh representation of the laser beam and the scanner head is constructed runtime. Out of the various geometric computation functions offered by PQP, collision detection relies on distance computation between pairs of objects. If the computed distance is smaller than a given threshold, then the two objects are declared colliding in a given robot position. Otherwise, the two objects do not collide. If, in a given robot position, none of the relevant pairs of objects collide, then the position itself is non-colliding.

Collision detection must ensure that the required minimum distance between the relevant pairs of objects is maintained while the robot moves along its *continuous path*. To provide this guarantee based on collision checks performed in an appropriately selected, finite set of *discrete positions*, the following method is applied. For each pair of relevant objects, a *lower tolerance* and an *upper tolerance* distance is introduced, denoted by d_l and d_u, respectively, with $d_l < d_u$. Collision checks in the selected positions are performed with a required minimum distance of d_u, which ensures that a minimum distance of d_l is maintained throughout the continuous path.

Let us denote by d^* the minimum distance of a given pair of objects along a continuous path. If $d^* < d_l$, then the above method classifies the path as colliding. If $d^* \geq d_u$, then the path is classified as non-colliding. However, if $d_l \leq d^* < d_u$, then the classification is undefined. Hence, parameter d_l specifies the minimum distance required between the objects, while d_u can be used to control the trade-off between geometric accuracy and computational efficiency (number of sample points required).

In the implemented collision detection method, separate tolerance parameters have been considered for the laser beam and scanner head, as shown in Table 1. Moreover, contact between the end of the laser beam and the workpiece is operational: this is the physical core of the welding process. Therefore, when performing collision detection between the laser beam and the workpiece, the beam length is truncated by e^L. No truncation is applied for collision detection against the fixture.

Finally, it is assumed that welding can be performed only when the complete stitch is visible from the laser emission point, and therefore, the theoretical possibility is ignored that portions of the stitch might become visible only gradually, as the scanner head moves along its path and welds other portions of the same stitch. This assumption is common in stitch welding (see, e.g., (Hatwig et al. 2012)).

Collision detection for a single robot position

A key procedure for collision-free path planing is collision detection for a given robot position, P. The definition of collision depends on the action performed in the given position:

Parameters for collision detection	
d_l^S	Lower tolerance distance for the scanner head
d_u^S	Upper tolerance distance for the scanner head
d_l^L	Lower tolerance distance for the laser beam
d_u^L	Upper tolerance distance for the laser beam
e^L	Laser beam end truncation
r^S	Radius of the scanner head model
Parameters for collision avoidance	
ϱ	Resolution of the 3D rectangular grid
B	Maximum bypass w.r.t. the original path
N	Neighborhood size for re-planning

Table 1: Parameters for collision detection and for collision avoidance.

when *welding a stitch*, both the scanner head and the laser beam are considered; during *idle movement*, the laser beam is switched off, and hence, only the scanner head is taken into account. The mesh models of the scanner head and the laser beam are constructed as follows:

Scanner head Since path planning precedes inverse kinematics in the proposed workflow, the orientation of the scanner head is unknown at the time of path planning. Hence, instead of a precise geometric model, the circumscribed sphere of the scanner head is used, which corresponds to a pessimistic assumption. Technically, this is achieved by using a mesh model that represents the scanner head as a single point P, and specifying $r^S + d_u^S$ as the distance threshold value in the PQP distance query.

Laser beam The mesh model of the laser beam for welding a *linear stitch* consists of a single triangle, as shown in Figure 5, corresponding to the assumption that the complete stitch is visible from the given robot position. The triangle is defined by the robot position (laser emission point), P, and the stitch start and end points, S_1 and S_2. In order to avoid false positive results near the workpiece, the height of the triangle is truncated by d_u^L when testing against the fixture, and by $d_u^L + e^L$ when testing against the workpiece. In both cases, a distance threshold of d_u^L is applied, resulting in the light gray collision zone for the fixture and the dark gray zone for the workpiece. In case of a *circular stitch* with radius r, the mesh consists of a single line between the laser emitting point and the stitch center point. The line is truncated by $d_u^L + r$ (fixture) or by $d_u^L + e^L + r$ (workpiece), and the distance threshold is set to $d_u^L + r$, resulting in a thin cylindrical volume that must be collision-free.

Collision detection for a continuous section

Collision detection is performed separately for each linear section of the broken line scanner path. Checking the linear section $\overline{P_1P_2}$ starts by collision detection for position $P = P_1$, and continues by checking subsequent discrete points of the section in the direction of P_2. The size of the discrete steps depends on the results of the distance queries, and it is chosen to guarantee that the prescribed lower tolerance

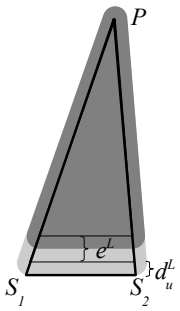


Figure 5: Mesh model of the laser beam for welding the linear stitch $\overline{S_1S_2}$ from robot position P. The approach results in the light gray collision zone for fixture, and the dark gray collision zone for both the fixture and the workpiece.

distance is maintained throughout the continuous path, even at points not directly checked. If all the checked positions are collision-free, then section $\overline{P_1P_2}$ itself is collision-free. Otherwise, the section is colliding. The pseudo-code of the algorithm is presented below.

```
PROCEDURE IsColliding(P1, P2)
    LET P := P1
    LOOP
        LET d := GetDistance(P)
        IF (d < du) THEN
            RETURN TRUE
        ELSE IF P = P2 THEN
            BREAK
        LET s := sqrt(d^2 - dl^2) + sqrt(du^2 - dl^2)
        IF d(P, P2) > s) THEN
            P := P + d(P1, P2) * s / d(P1, P2)
        ELSE
            P := P2
    RETURN FALSE
```

In the pseudo-code, function GetDistance(P) executes a PQP distance query for the single robot position P. The tolerance distance parameters d_l and d_u are set as presented above. The correctness of the procedure is proven in the following lemma, focusing on two subsequent robot positions investigated in the inner loop of the algorithm, denoted as P and P', for collisions of the scanner head, represented as a single point mesh model.

Lemma 1 *Let P and P' be two points in space such that their shortest distance from a given, fixed 3D object O is $d(P, O) = d \geq d_u$ and $d(P', O) \geq d_u$. Now, if $d(P, P') \leq \sqrt{d^2 - d_l^2} + \sqrt{d_u^2 - d_l^2} = s$, then for any point Q of section $\overline{PP'}$, it holds that $d(Q, O) \geq d_l$.*

Proof. Assume that the shortest distance between the object O and the section $\overline{PP'}$ arises between points $R \in O$ and $Q \in \overline{PP'}$ (see Figure 6). If $Q = P$ or $Q = P'$ then the lemma is trivial. Otherwise, Q is an internal point of section $\overline{PP'}$. If $d(P, P') \leq \sqrt{d^2 - d_l^2} + \sqrt{d_u^2 - d_l^2}$, then either $d(P, Q) \leq \sqrt{d^2 - d_l^2}$ or $d(Q, P') \leq \sqrt{d_u^2 - d_l^2}$. Assume that the first case holds. Then, PQR is a right triangle with

hypotenuse d. By the Pythagorean theorem, if $d(P,Q) \leq \sqrt{d^2 - d_l^2}$, then $d(Q,R) \geq l_2$, and the lemma is proven. For the second case, similar claims can be made for the triangle $P'QR$. □

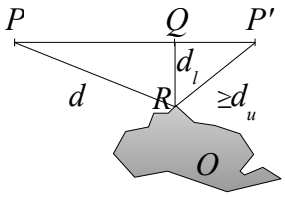


Figure 6: Illustration of the proof of the correctness of the procedure for collision checking on the continuous section $\overline{PP'}$.

It is straightforward to generalize the lemma to the laser beam as well. The proof exploits that the mesh model of the beam consists of triangles with one vertex corresponding to the laser emission point, and two (possibly coinciding) vertices are fixed while the robot moves along its path. Each point of such a triangle moves along a linear section as the laser emission point moves along $\overline{PP'}$.

Collision-free path planning

Representation of the path

It is assumed that the potentially colliding rough-cut path is described as a list $((P_1, a_1), (P_2, a_2), ..., (P_k, a_k))$, where segment (P_i, a_i) denotes that the robot moves from point P_i to point P_{i+1} along a linear section while performing action a_i. Action a_i can be of two types: $a_i = (s_{[i]}, +)$ or $a_i = (s_{[i]}, -)$. Action $a_i = (s_{[i]}, +)$ encodes welding stitch $s_{[i]}$, where $s_{[i]}$ corresponds to one of the stitches $s_1, ..., s_n$, sequenced to the ith position of the path. In contrast, $a_i = (s_{[i]}, -)$ denotes idle movement directly after welding $s_{[i]}$. The rough-cut path contains exactly one segment for welding each stitch, and zero or one idle movement segment between two welding segments, hence, $n \leq k < 2n$. Note that the same does not hold for the collision-free path, since it might be necessary to move the robot along a more complex path to avoid collisions, both while welding and during idle movement.

It is assumed that each segment (P_i, a_i) is labeled as colliding or non-colliding by the above collision detection procedure. Collision avoidance relaxes the colliding segments of the path, as well as the segments that are close to colliding segments. More specifically, segment (P_i, a_i) is relaxed if and only if there exists j such that $i - N \leq j \leq i + N$ and segment (P_j, a_j) is colliding, where N is the neighborhood size for re-planning. The procedure is illustrated in Figure 7, where the colliding segments (red) and their neighborhood with $N = 1$ are re-planned, resulting in a collision-free path (blue).

As a result, the rough-cut path consists of a series of relaxed and non-relaxed segments. Collision avoidance is performed on maximal relaxed sections of the path, and replaces these relaxed sections by new, collision-free sections.

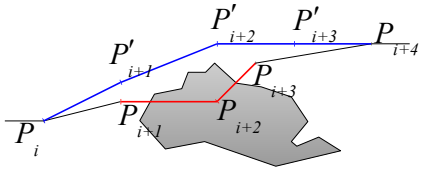


Figure 7: Collision avoidance by replanning the colliding segments (red) and their neighborhood with $N = 1$.

The proposed procedure preserves the order of the stitches, but it may modify the number of segments in the path, as well as the points visited along the path.

In the sequel, we assume that collision avoidance is performed for a single, maximal relaxed section of the rough-cut path, $((P_\alpha, a_\alpha), (P_{\alpha+1}, a_{\alpha+1}), ..., (P_\beta, a_\beta))$. Furthermore, let $(s_{\{1\}}, s_{\{2\}}, ..., s_{\{m\}})$ denote the sequence of welding the stitches along the relaxed path section. If there are several, disjoint relaxed sections to re-plan, then the same procedure is repeated on each of those sections.

Representation of the collision map

The state space for collision-free path planning is represented as a four-dimensional map of discrete vertices, with the three spacial dimensions and one additional dimension describing the action performed in the vertex. The map contains the combination of a 3D point and an action, $(P, a = (s, +))$ as a vertex if and only if P is contained in the CFAV of stitch s. The pair $(P, a = (s, -))$ is contained in the map if P itself is collision-free (with the laser beam switched off), i.e., $P \in \mathrm{CF}_0$.

The points included in the map are the points of a discretized, rectangular 3D grid with a resolution of $\varrho = \min(d_u^S - d_l^S, d_u^L - d_l^L)$. By Lemma 1, the application of this resolution and collision checks in the grid points with tolerance d_u^S and d_u^L ensure that movement between two neighboring grid points is collision-free with d_l^S and d_l^L. The map is created for a finite rectangular area, defined by values $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, $z_{\min}$, and $z_{\max}$, where $x_{\min} = \min_{i=\alpha}^{\beta} x(P_i) - B$ and B is the maximum bypass parameter, and other boundary parameters are computed analogously.

Possible transitions between states are captured by directed arcs between the vertices, according to the following rules. Let $N(P)$ denote the 6-neighborhood of point P, i.e., the set of six neighboring points along the x, y, and z axis. From the vertex capturing action $(s_{\{i\}}, +)$ in P, there are arcs to

- $(s_{\{i\}}, +)$ in $N(P)$, i.e., continuing the welding operation in a neighboring point;
- $(s_{\{i\}}, -)$ in $N(P)$, i.e., finishing the welding operation and continuing with idle movement;
- $(s_{\{i+1\}}, +)$ in $N(P) \cup P$, i.e., continuing with welding the next stitch.

From the vertex encoding $(s_{\{i\}}, -)$ in P, there are arcs to

- $(s_{\{i\}}, -)$ in $N(P)$, i.e., continuing the idle movement in one of the neighboring points;

- $(s_{\{i+1\}}, +)$ in $N(P)$, i.e., welding the next stitch.

To save computation time by omitting unnecessary collision checks, the proposed procedure does not generate the complete collision map at once. Instead, vertices are generated and checked for collisions on the fly, as they are explored by the search procedure. Moreover, the results of collision detection are inferred from the results for the neighboring points whenever possible.

A* search for a collision-free path

In order to compute a collision-free path, an A* search is performed on the above defined collision map. Each node of the search tree is represented as a tuple $\Gamma = (P, a, r, t)$, where P is a 3D point and a is an action, corresponding to a vertex in the collision map. The non-negative real r is the time remaining for welding stitch corresponding to a, and t is the total time of traveling the path from the source node to Γ. Note that if a is an idle movement action, then $r = 0$.

The source node of the search is defined as $(P_\alpha, (s_{\{1\}}, +), t_{\{1\}}, 0)$, and goal states are of the form $(P_\beta, (s_{\{m\}}, +), 0, \cdot)$. A special case arises when the relaxed section is at the beginning of the rough-cut path, since in this case, the collision-free path can start at any point P in $\text{CFAV}_{\{1\}}$. Accordingly, search is initialized with multiple source nodes in the list of open nodes, one for each such point P. Similarly, when the relaxed section is at the end of the rough-cut path, then it can terminate anywhere in $\text{CFAV}_{\{m\}}$.

The cost function of the A* search is t, while the heuristic function h is a lower estimate of the remaining time. In a node $\Gamma = (P, (s_{\{i\}}, \cdot), r, t)$, the heuristic value is computed as

$$h(\Gamma) \;=\; \max\left(r + \sum_{j=i+1}^{m} t_{\{j\}},\; \frac{d(P, P_\beta)}{v}\right).$$

The first term encodes the total remaining welding time on the current stitch and on the future stitches. The second term is the time for traveling from the current location to the goal point P_β. When there are multiple goal points, the second term is ignored.

According to the rules of the A* search, in each step, a node with minimal $t + h$ is expanded. When expanding a node $\Gamma = (P, a, r, t)$, Γ is removed from the open list, and a new node $\Gamma' = (P', a', r', t')$ is created and inserted into the list of open nodes for each directed neighbor of (P, a) in the collision map. The new node inherits P' and a' from the vertex of the map, whereas parameters r and t are computed as follows.

- If a' is welding the same stitch from a different position, then $r' = \max(r - \frac{\varrho}{v}, 0)$ and $t' = t + \frac{\varrho}{v}$;
- If a' is welding the subsequent stitch and $P = P'$, then $r' = t_{\{i+1\}}$ and $t' = t + r$;
- If a' is welding the subsequent stitch and $P \neq P'$, then $r' = t_{\{i+1\}}$ and $t' = t + \max(\frac{\varrho}{v}, r)$;
- If a' is idle movement, then $r' = 0$ and $t' = t + \max(\frac{\varrho}{v}, r)$;

This search step is iterated until a goal state is reached. Links between nodes and their parents are maintained throughout the search, and sequence of links from the first goal state to the source state encodes a collision-free path from P_α to P_β.

Let there be given two search nodes belonging to the same point P, denoted by $\Gamma = (P, a, r, t)$ and $\Gamma' = (P, a', r', t')$. The following two dominance rules are defined.

Dominance rule #1: If $t < t'$ and $t + h(\Gamma) < t' + h(\Gamma')$, then let Γ' be fathomed.

Dominance rule #2: If $t < t'$, then let Γ' be fathomed.

While rule #1 is obviously admissible, the stronger rule #2 is an inadmissible dominance rule, and may result in losing the optimal collision-free path. However, even the application of rule #2 maintains the completeness of the search, i.e., it is guaranteed that a feasible collision-free path is found if there exists one. In our implementation we have decided to apply rule #2, since initial experiments we have found that it brings considerable speed-up with negligible loss of performance.

Smoothing the path

The path computed by the A^* search consists of small, axial sections in the Cartesian coordinate system of the workpiece. This path is smoothed by eliminating the unnecessary breakpoints using an algorithm that considers each section (P_i, a_i) one-by-one. If $a_i \equiv a_{i-1}$ and section $\overline{P_{i-1}P_{i+1}}$ is collision-free for executing action a_i, then this section is removed from the path, which implicitly entails that the previous section (P_{i-1}, a_{i-1}) is extended until point P_{i+1}. The procedure is illustrated in Figure 8. Finally, the smoothed collision-free path segments are inserted at the place of the removed, colliding path segments, and the cycle time is recalculated.

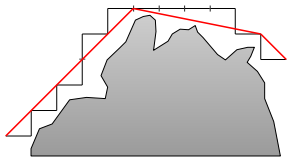


Figure 8: Comparison of the initial (black) and the smoothed (red) collision-free paths.

Experimental Results

Comparison of Different Algorithms

The proposed algorithms have been evaluated on problems involving the assembly of a car front door using RLW. Experiments have been performed on real industrial data, containing a single door geometry with different stitch layouts, various fixture designs, and realistic technological parameters. The instances contained 28-71 welding stitches. The mesh model of the door geometry consisted of ca. 10^5 triangles, while the fixture model contained $5 \cdot 10^5$ triangles.

The experiments involved computing a task sequence and a rough-cut path by three different algorithms for each instance, and converting all the three solutions to a collision-

free path by the algorithm proposed above. The three sequencing algorithms are as follows:

- TS-PP, our algorithm for integrated task sequencing and path planning (Kovács 2013);
- RMV, the single sequencing algorithm dedicated to RLW from the literature (Reinhart, Munzert, and Vogl 2008), which solves a TSP over the stitch positions. Hence, this algorithm focuses on the length of the tool contact point (TCP) path when optimizing the stitch sequence;
- RMV*, a modified version of RMV that solves the TSP over the mid-points of the access volumes, instead of the stitch position. This modification implies that RMV* addresses the minimization of the length of the scanner path, instead of the TCP path.

All algorithms have been implemented in C++. RMV and RMV* used ILOG CP as a TSP solver. The experiments were run on a 2.66 GHz Intel Core 2 Duo computer. A time limit of 120 seconds was applied.

The proposed algorithms computed a feasible, collision-free robot path for every instance with all the three task sequencing methods. The results unambiguously indicate the dominance of robot path planning (TS-PP and RMV*) over TCP path planning approaches, see Figure 9. For workpieces with complex geometry, RMV leads to moving the scanner head in a zigzag above stitches that have nearby positions but different surface normals. In case of a car door, this phenomenon is the most spectacular around the window frame, where the stitches on the inner and the outer sides are close to each other, but must be welded from opposite directions. Consequently, in our experiments, RMV resulted in up to 3 times higher cycle times and up to 15 times higher idle times than TT-PS.

The detailed comparison of the three algorithms is presented in Table 2, where each row stands for a separate problem instance. Instance names beginning with W and WF refer to welding without fixture and with fixture, respectively. Column n contains the number of stitches, while *min. accessibility* and *avg. accessibility* present the minimum and average accessibility ratio, i.e., the ratio of CFAV and TAV, measured over the different stitches in percent. For each algorithm, columns *cycle1* and *cycle2* contain the cycle time of the rough-cut path and the collision-free path, respectively. The best cycle times are denoted by bold font for each instance. Columns *run* contain the run time of the algorithm in seconds. It is noted that for RMV and RMV*, the TSP solver terminated with a locally optimal sequence in less then 1 second, hence, *run* is practically the time required for collision avoidance. In contrast, TS-PP was run for 120 seconds on each instance, plus the time of collision avoidance.

The results show a notable difference among the instances depending on stitch accessibility. For the WF instances, accessibility was very poor (minimum accessibility around 10%, average accessibility of 60-70%). For the W instances, collision avoidance was run with the workpiece geometry only, resulting in 24-50% min. and around 90% average accessibility. The reason of poor accessibility with fixture was twofold. First, the car door was originally designed for spot welding, and the stitch layout was received by replacing the spots by RLW stitches, with minor modifications; in fact, ca. 20-40% less stitches could ensure sufficient stiffness. On the other hand, the key design objective for the experimental fixture was to achieve perfect gap control, while the general design guideline that the stitch accessibility volumes must be kept clear was ignored. It is noted that several instances had to be pre-processed to eliminate stitches that are completely inaccessible, since otherwise the path planning problem would have no feasible solution. After all, we expect that for a car door in production, stitch accessibility and the complexity of collision avoidance would be somewhere between those experienced for the W and the WF instances.

Regarding algorithm performance, TS-PP reduced cycle times drastically compared to RMV. The reduction was on average 63% on the rough-cut path, and 61% on the collision-free path. This was mostly due to the joint consideration of the TCP and the scanner movement, instead of optimizing the TCP path only.

TS-PP also outperformed RMV* regarding the cycle time of the rough-cut path on every instance, by computing up to 6.1%, on average 2.9% more efficient paths. However, this did not automatically translate to improvement on the collision-free path on each individual instance. The perturbation of the rough-cut paths by collision avoidance resulted in a situation where TS-PP computed better collision-free paths on 11 out of 15 instances, by up to 4.3%. However, RMV* outperformed TS-PP on 4 of the 15 instances, by 2.1-4.9% on the different instances. This occurred typically for the WF instances with the worst accessibility. Beyond the random perturbation caused by the modifications to the rough-cut path, a possible explanation of this phenomenon comes from the different underlying assumptions made by the algorithms for sequencing. Implicitly, RMV* assumes that each stitch is welded from the mid-point of the technological access volume, whereas TS-PP assumes that the complete technological access volume can be used. In these problematic instances, the assumption of RMV* appears to be closer to reality. Initial experiments on sequencing using reduced TAVs confirm this hypothesis, and with an appropriate choice of parameters, the method resulted in TS-PP outperforming RMV* an all instance, but an elaborate heuristic is subject to future work.

The average computation time was 165 seconds and 119 seconds for RMV and RMV*. TS-PP required 241 seconds on average, due to the higher computation time of sequencing. Half of the computation time was taken by sequencing and rough-cut path planning, while the other half by collision-free path planning. Still, these response times comply with industrial expectations, and enable the use of the algorithms in a decision support tool in an iterative design and planning process.

Conclusions

This paper introduced a new collision-free path planning algorithm for RLW. The algorithm departs from a task sequence and a potentially colliding rough-cut path, and alters this path to achieve a collision-free path with minimal

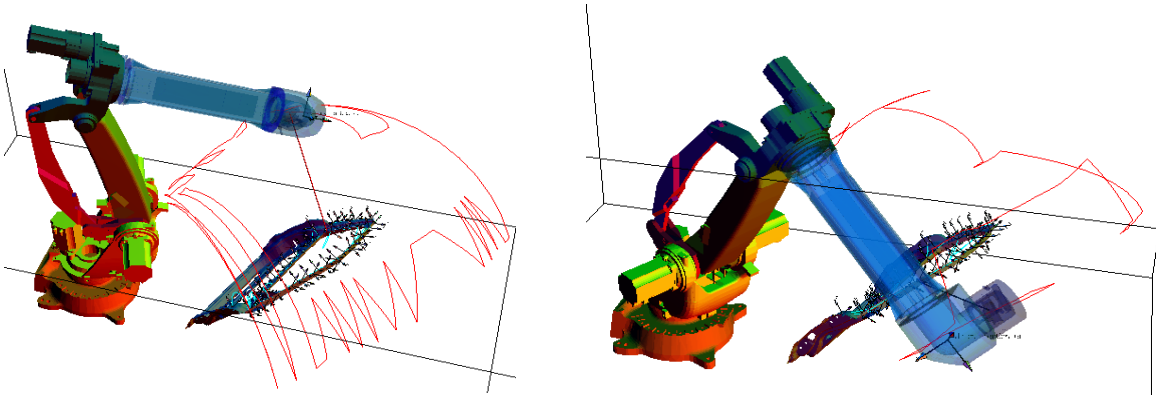


Figure 9: Comparison of the paths computed by the RMV (left) and the proposed TS-PP (right) methods. TS-PP focuses on the scanner path, and geometrical and technological parameters already at the time of task sequencing, which results in shorter scanner path and reduced cycle time.

		Accessibility		RMV			RMV*			TS-PP		
	n	min.	avg.	cycle1	cycle2	run	cycle1	cycle2	run	cycle1	cycle2	run
W1	28	47.32	91.63	30.05	30.53	22	14.01	14.01	7	**13.69**	**13.69**	128
W2	34	47.32	95.16	35.50	35.50	2	15.93	15.93	2	**15.48**	**15.49**	135
W3	62	49.29	93.61	76.39	76.64	11	26.91	26.91	2	**26.11**	**26.11**	122
W4	44	34.38	87.21	56.33	57.23	19	19.55	19.84	8	**18.36**	**18.98**	146
W5	71	24.46	90.76	78.64	78.64	3	30.29	30.29	3	**29.85**	**29.85**	123
W6	67	24.46	90.84	67.70	67.70	3	28.50	28.50	3	**27.75**	**27.75**	123
WF1	28	10.97	64.21	30.05	31.26	229	14.01	15.04	113	**13.69**	**14.69**	299
WF2	34	14.63	69.49	35.50	35.86	286	15.93	16.92	192	**15.48**	**16.42**	337
WF3	62	11.14	68.49	76.39	78.42	294	26.91	**27.51**	219	**26.11**	28.08	353
WF4	44	10.89	58.81	56.33	58.23	163	19.55	21.16	196	**18.37**	**21.02**	283
WF5	64	9.79	65.03	75.37	77.18	270	26.15	**27.58**	249	**26.10**	28.93	448
WF6	63	9.79	63.55	74.92	76.40	399	25.81	**27.25**	365	**25.07**	27.91	404
WF7	63	6.90	60.98	74.92	76.59	504	25.81	**27.38**	334	**25.07**	27.95	421
Avg.	51	21.04	72.18	59.08	60.01	170	22.26	22.95	130	**21.63**	**22.84**	256

Table 2: Comparison of the RMV, RMV*, and the proposed TS-PP algorithms.

cycle time by iterating shortest path algorithms and distance queries on a mesh model representation of the involved moving objects. Extensive computational experiments have shown that the proposed algorithms are efficient in solving real industrial problems originating from the automotive industry.

Nevertheless, the results achieved permit drawing conclusions in a wider context as well. Most importantly, it has been shown that in RLW, and in general, for machining technologies where relatively slow robot motion is coupled with quick movements of the tool, optimization must jointly consider the robot path and the tool path, instead of focusing solely on the tool path. For the car door designs considered in our experiments, this resulted in an enormous reduction of the cycle times, by 63% for on average.

Second, while tool positions are well defined for the effective tasks, e.g., stitch positions in RLW, one has a significant degree of freedom in choosing the corresponding robot path. On the one hand, this freedom opens new opportunities for optimization, but on the other hand, it presents a serious computational challenge, and an efficient combination of combinatorial optimization and geometric reasoning is required for tackling it. While most earlier contributions applied a sampling strategy to solve sequencing and path planning over a finite set of pre-defined discrete points, we proposed algorithms for planning in the continuous space, using efficient geometric computation routines.

Our current research focuses on improving the stitch sequence and the rough-cut path on instances with poor accessibility, by heuristics that adjust the technological access volumes to the real, collision-free access volumes. Furthermore, the verification and thorough evaluation of the developed off-line programming toolbox in physical experiments is underway.

Acknowledgements

The author thanks József Váncza and Gábor Erdős for the helpful discussions. This work has been supported by EU FP7 grant RLW Navigator No. 285051 and the NFÜ grant ED-13-2-2013-0002.

References

Alatartsev, S.; Augustine, M.; and Ortmeier, F. 2013. Constricting insertion heuristic for traveling salesman problem with neighborhoods. In *Proc. of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-2013)*, 2–10.

Alatartsev, S.; Mersheeva, V.; Augustine, M.; and Ortmeier, F. 2013. On optimizing a sequence of robotic tasks. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2013)*, 217–223.

Castelino, K.; D'Souza, R.; and Wright, P. K. 2002. Toolpath optimization for minimizing airtime during machining. *Journal of Manufacturing Systems* 22(3):173–180.

Dewil, R.; Vansteenwegen, P.; and Cattrysse, D. 2014. Construction heuristics for generating tool paths for laser cutters. *International Journal of Production Research* in print.

Erdős, G.; Kemény, Z.; Kovács, A.; and Váncza, J. 2013. Planning of remote laser welding processes. *Procedia CIRP* 7:222–227.

Ferguson, D., and Stentz, A. 2006. Using interpolation to improve path planning: The field D* algorithm. *Journal of Field Robotics* 23(2):79–101.

Geraerts, R., and Overmars, M. H. 2002. A comparative study of probabilistic roadmap planners. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, 43–57.

Hatwig, J.; Minnerup, P.; Zaeh, M. F.; and Reinhart, G. 2012. An automated path planning system for a robot with a laser scanner for remote laser cutting and welding. In *2012 IEEE International Conference on Mechatronics and Automation (ICMA)*, 1323–1328.

Hatwig, J.; Reinhart, G.; and Zaeh, M. F. 2010. Automated task planning for industrial robots and laser scanners for remote laser beam welding and cutting. *Production Engineering* 4(4):327–332.

Hudson, T. C.; Lin, M. C.; Cohen, J.; Gottschalk, S.; and Manocha, D. 1997. V-collide: Accelerated collision detection for vrml. In *VRML 97: Second Symposium on the Virtual Reality Modeling Language*, 119–125.

Kaelbling, L., and Lozano-Perez, T. 2011. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 1470–1477.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

Koenig, S., and Likhachev, M. 2005. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics* 21(3):354–363.

Kolakowska, E.; Smith, S. F.; and Kristiansen, M. 2014. Constraint optimization model of a scheduling problem for a robotic arm in automatic systems. *Robotics and Autonomous Systems* 62(2):267–280.

Kovács, A. 2013. Task sequencing for remote laser welding in the automotive industry. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-2013)*, 457–461.

Kucuk, S., and Bingul, Z. 2006. Robot kinematics: Forward and inverse kinematics. In Cubero, S., ed., *Industrial Robotics: Theory, Modelling and Control*. Pro Literatur Verlag. 117–148.

Kuffner, J. J., and LaValle, S. M. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'00)*, 995–1001.

Larsen, E.; Gottschalk, S.; Lin, M. C.; and Manocha, D. 2000. Fast proximity queries with swept sphere volumes. In *Proc. IEEE Int. Conf. Robot. Autom.*, 3719–3726.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS)*.

Park, H.-S., and Choi, H.-W. 2010. Development of digital laser welding system for car side panels. In Na, X., ed., *Laser Welding*. InTech. 181–192.

Park, C.; Pan, J.; Lin, M.; and Manocha, D. 2013. Realtime gpu-based motion planning for task execution in dynamic environments. In *Proceedings of the 1st Workshop on Planning and Robotics (PlanRob 2013)*, 60–63.

Reinhart, G.; Munzert, U.; and Vogl, W. 2008. A programming system for robot-based remote-laser-welding with conventional optics. *CIRP Annals – Manufacturing Technology* 57(1):37–40.

Saha, M.; Sánchez-Ante, G.; Roughgarden, T.; and Latombe, J.-C. 2006. Planning tours of robotic arms among partitioned goals. *International Journal of Robotics Research* 25(3):207–223.

Shibata, K. 2008. Recent automotive applications of laser processing in Japan. *The Review of Laser Engineering* 36:1188–1191.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*.

Stentz, A. 1995. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1652–1659.

Trenkel, S.; Weller, R.; and Zachmann, G. 2007. A benchmarking suite for static collision detection algorithms. In *International Conference in Central Europe on computer graphics, visualization and computer vision (WSCG)*.

Tsoukantas, G.; Salonitis, K.; Stournaras, A.; Stavropoulos, P.; and Chryssolouris, G. 2007. On optical design limitations of generalized two-mirror remote beam delivery laser systems: the case of remote welding. *The International Journal of Advanced Manufacturing Technology* 32(9–10):932–941.

Author Index

Aha, David	106
Aksakalli, Vural	166
Alami, Rachid	20
Alford, Ronald	106
Alkaya, Ali Fuat	166
Amato, Christopher	69
Apker, Thomas	106
Arras, Kai Oliver	90, 136
Auslander, Bryan	106
Awaad, Iman	117
Beck, Chris	59
Bekris, Kostas	80
Bit-Monnot, Arthur	12
Borgo, Stefano	28
Brafman, Ronen	99
Cesta, Amedeo	28
Chaudhuri, Swarat	145
Cruz, Gabriel	69
de Silva, Lavindra	20
Dvorak, Fiip	12
Garrett, Caelan	148
Gaschler, Andre	157
Gelfond, Michael	127
Ghallab, Malik	12
Hertzberg, Joachim	117
How, Jonathan	69
Infantes, Guillaume	49
Ingrand, Félix	12
Kaelbling, Leslie	69, 148
Karneeb, Justin	106
Kavraki, Lydia	145
Kiesel, Scott	1
Kimmel, Andrew	80
Konidaris, George	69
Kovacs, Andras	172
Kraetzschmar, Gerhard	117

Lallement, Raphaël 20
Lesire, Charles 49
Lozano-Perez, Tomas 148

Maynor, Christopher 69
McMahon, James 106
Molineaux, Matthew 106
Moll, Mark 145

Nedunuri, Srinivas 145
Nejat, Goldie 59, 136

Orlandini, Andrea 28

Palmieri, Luigi 90
Petrick, Ron 157
Prabhu, Sailesh 145
Pralet, Cédric 49

Rasconi, Riccardo 28
Roberts, Mark 106
Ruml, Wheeler 1

Sahin, Furkan 166
Scala, Enrico 38
Schwenk, Markus 136
Shani, Guy 99
Shimony, Solomon 99
Sridharan, Mohan 127
Suriano, Marco 28

Umbrico, Alessandro 28

Vaquero, Tiago Stegun 59, 136
Vattam, Swaroop 106

Wilson, Mark 106
Wyatt, Jeremy 127

Zhang, Shiqi 127