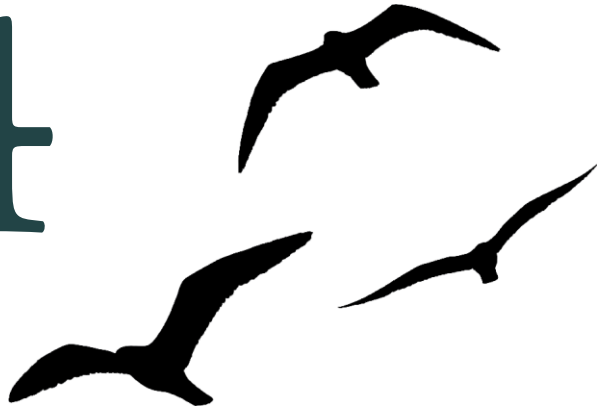
ICAPS 2014



Proceedings of the 2nd Workshop on

Distributed and Multi-Agent Planning

Edited By:

Daniel Borrajo, Daniel L. Kovacs and Alejandro Torreño.

Portsmouth, New Hampshire, USA - June 22, 2014

DISTRIBUTED AND MULTI-AGENT PLANNING

Organizing Committee

Daniel Borrajo
Universidad Carlos III de Madrid, Spain
Daniel L. Kovacs
Budapest University of Technology and Economics, Hungary
Alejandro Torreño
Universitat Politècnica de València, Spain

Program Committee

Ronen Brafman, Ben-Gurion University of the Negev, Israel
Bradley J. Clement, NASA Jet Propulsion Laboratory, USA
Amanda Coles, King's College London, UK
Tadeusz P. Dobrowiecki, Budapest University of Technology and Econonmics, Hungary
Naoki Fukuta, Shizuoka University, Japan
Antonín Komenda, Technion, Israel Institute of Technology, Israel
Raz Nissim, Ben-Gurion University of the Negev, Israel
Eva Onaindia, Universidad Politècnica de València, Spain
Scott Sanner, National ICT Australia, NICTA, Australia
Matthijs Spaan, Delft University of Technology, Netherlands
Roni Stern, Harvard University, USA
Mathijs de Weerdt, Delft University of Technology, Netherlands
Shlomo Zilberstein, University of Massachusetts, Amherst, USA

Foreword

This volume compiles the scientific papers accepted at DMAP'14, the 2nd Distributed and Multi-Agent Planning workshop, held at ICAPS 2014 in Portsmouth, New Hampshire, USA, on June 22nd, 2014. This event follows up the successful first edition of DMAP, held at ICAPS 2013, and continues the tradition of the "Multiagent Planning and Scheduling" workshop series held at ICAPS 2005 and 2008, and the joint AAMAS-ICAPS session on multi-agent planning in 2010.

Distributed and Multi-Agent Planning is a broad field of growing interest within the planning and scheduling community. However, its subfields remain dispersed and uncoordinated. Most works in this field are generally published in major AI conferences, such as IJCAI, and AAAI, AAMAS or ICAPS. Nevertheless, most of these approaches are based or built upon planning and scheduling technologies developed by the ICAPS community.

The aim of the workshop is to bring together the multi-agent planning community, offering researchers a forum to introduce and discuss their works in the different subfields of this area that do not have room in the main ICAPS conference. Although, this year authors were given the opportunity to introduce their works also in the poster session of the main ICAPS conference.

This year's papers cover many of topics of multi-agent planning, such as heuristics, negotiation, cooperation, beliefs, game theory, preference-based planning, privacy-preservation and multi-agent plan merging, among others. Therefore, the workshop offers a wide view of the state-of-the-art in multi-agent planning, fostering works that have the potential to push this field forward.

Altogether 12 submissions were received from 6 different countries, 11 full papers and 1 short position paper, matching the numbers of the first edition of the workshop. These papers were reviewed by a Program Committee composed of 13 members and coordinated by the 3 PC chairs. As a result of the review process, all 11 full papers were accepted and will be presented orally at the workshop.

We thank the members of the Program Committee for their dedicated effort at reviewing and ensuring the quality of the works presented at DMAP'14. We also thank the chairs of the ICAPS main conference for their continuous support throughout the organization of this workshop. Finally, we would like to thank our authors and the multi-agent planning community for submitting their work to DMAP'14. Your research, collaboration and active participation are critical to the success of this workshop.

– Daniel Borrajo, Daniel L. Kovacs, Alejandro Torreño
DMAP-2014 Chairs

Table of Contents

Distributed Heuristic Forward Search for Multi-Agent Systems

Ronen Brafman and **Uri Zoran**
Ben-Gurion University of the Negev
Be'er Sheva, Israel
brafman,zoranu@cs.bgu.ac.il

Abstract

We consider possible extensions to MA-STRIPS and its algorithms. Our focus is on modeling and planning with interacting actions: we show how to extend Nissim and Brafman's multi-agent forward search algorithm to this setting. We also discuss a few future challenges: task allocation problems and finer-grained distinctions between actions and variables.

Introduction

There is a long tradition of work on multi-agent planning for cooperative and non-cooperative agent teams involving centralized and distributed algorithms, often using involved models that model uncertainty, resources, and more (Szer, Charpillet, and Zilberstein 2005), and much work on how to coordinate local plans of agents or to allow agents to plan locally under constraints (Cox and Durfee 2005; Steenhuisen et al. 2006; ter Mors and Witteveen 2005). Recently, Brafman and Domshlak (Brafman and Domshlak 2008) started exploring a more modest, minimalist model of multi-agent planning using MA-STRIPS or similar language. We shall refer to this as *classical multi-agent planning*. Although originally this work was motivated by the goal of exploiting multi-agent structure within single-agent domains, later work, such as (Nissim, Brafman, and Domshlak 2010; Torreño, Onaindia, and Sapena 2012), used this framework to study to distributed planning by a group of cooperating agents, and in particular, privacy preserving algorithms that allow agents to cooperate without revealing their private information.

MA-STRIPS extends STRIPS by associating each action with an agent. This is probably the minimal extension possible to introduce elements of multi-agent systems. MA-STRIPS took this minimalist approach for two reasons: First, in order to be as similar to classical planning and understand the essential difference from single-agent planning, allowing us to import, as much as possible, some of the powerful techniques developed in classical single agent planning. Second, in order to allow us to study multi-agent planning in its simplest form, as it is often easier to formulate new ideas within simple models. Unfortunately, MA-STRIPS does not allow us to model a fundamental aspect of many multi-agent systems: interacting actions.

Interacting actions are actions whose combined effect is ill-defined or different from the union of effects of each action, such as actions with conflicting effects. Such cases are easily handled by disallowing concurrent actions. Indeed, recent algorithms for classical multi-agent planning essentially assuming that actions are performed at different time points. Later on, one can try to parallelize plan execution in order to reduce the plan's execution time (makespan), by allowing the concurrent non-interacting actions. A more interesting case is that of actions with synergetic effects. For example, a single agent may not be able to push a heavy box, but two agents acting simultaneously can.

Interacting actions raise two issues: representation and planning. Boutilier and Brafman (Boutilier and Brafman 2001) proposed a simple adaptation of STRIPS that can express action interaction, and a non-privacy preserving partial-order planning algorithm for such problems. It is quite possible that the MA-POP algorithm (Torreño, Onaindia, and Sapena 2012) could be extended to a privacy preserving version for such problems by combining it with the ideas of Boutilier and Brafman. However, in this paper we explain how a simple change to the multi-agent forward search (MAFS) schema of Nissim and Brafman (Nissim and Brafman 2013a; 2012) can support this extension. MAFS has two advantages over MA-POP: it is much faster, and it can generate optimal plans when instantiated using A^*. We note that a related method for representing joint actions was described in (Kovacs 2012).

In addition to the issue of interacting actions, we also discuss two topics that come up in the context of MA-STRIPS which we believe are interesting to explore. The first is the definition of private and public variables and actions, and the second is that of identical actions belonging to different agents. The first issue can lead to more refined definitions, and consequently, more efficient algorithms, while the latter provides an opportunity to address the issue of task allocation with the simple MA-STRIPS model.

Background

MA-STRIPS

A MA-STRIPS problem for a set of agents $\Phi = \{\varphi_i\}_{i=1}^k$ is given by a 4-tuple $\Pi = \langle P, \{A_i\}_{i=1}^k, I, G\rangle$, where P is a finite set of propositions, $I \subseteq P$ and $G \subseteq P$ encode the initial state and goal, respectively, and for $1 \leq i \leq k$, A_i is the set of actions agent φ_i is capable of performing. Each action $a = \langle \mathrm{pre}(a), \mathrm{eff}(a)\rangle$ is given by its preconditions and effects. A plan is a solution to Π iff it is a solution to the underlying STRIPS problem obtained by ignoring the identities of the agent associated with each action. Since each action is associated with an agent, a plan tells each agent what to do and when. In different planning contexts, one might seeks special types of solutions. For example, in the context of planning games (Brafman et al. 2009), *stable* solutions are sought. We focus on cooperative multi-agent systems, seeking either a (standard) solution or a cost-optimal solution.

The partitioning of the actions to agents yields a distinction between private and public propositions and actions. A *private* proposition of agent φ is required and affected only by the actions of φ. An action is *private* if all its preconditions and effects are private. All other actions are classified as *public*. That is, φ's private actions affect and are affected only by φ's actions, while its public actions may require or affect the actions of other agents.

If privacy is not an issue, the distinction between private and public actions is not essential, although it can be exploited for computational gains (Brafman and Domshlak 2008). But in some settings, agents collaborate on a specific task, yet prefer not to reveal private information about their local state, their private actions, and their cost. We refer to algorithms that plan without revealing this information as *privacy preserving* . In a privacy-preserving algorithm the only information available about an agent to others is its set of public actions, projected onto public atoms. This can be viewed as the interfaces between the agents. Information about an agent's private actions and private aspects of a public action are known to the agent only.

Interacting Actions

To the best of our knowledge, Boutilier and Brafman's work (BB) (Boutilier and Brafman 2001) was the first extension of STRIPS-like languages to address interacting actions, and the first to propose a planner that can handle such domains. Their extension to STRIPS is conceptually simple: in addition to a list of preconditions, add to the description of an action a a concurrency condition that specifies which actions must or must not be executed concurrently with a without affecting a's effects. Conditional effects are changed similarly, allowing the effect condition to specify concurrent actions that influence the conditional effect.

As an example, consider an action for lifting the side of the table. If performed by two agents on both sides, objects on the table will remain. But if performed by a single agent, the objects will fall. Thus, the action of lifting the left side of a table will have, beyond its regular preconditions and effects, a conditional effect with concurrent effect condition that states that the action of lifting the right side is not performed concurrently, and a conditional effect that objects on the table are not longer on the table. Similarly, a table can be moved only if both agents holding its side move in the same direction. Thus, the action of moving the table north by one agent will have a concurrency condition that requires a move-north action by another agent. Naturally, that action will have a precondition that that agent is actually holding the table. A similar, third, example is box pushing – if the box is heavy and one agent pushes it, it remains in place. If two agents push it then it will move. Thus, the box movement is a conditional effect with a concurrency condition requiring another push action.

Interacting Actions in MA-STRIPS

We see two possible ways of extending MA-STRIPS to address interacting actions. The first involves adapting the syntax of BB as is. In that case, what is being changed is the underlying STRIPS language, and the extension of MA-STRIPS is identical – requiring us to associate an agent with each action. A second possibility is to move to a model that uses joint actions. A joint-action is an action executed concurrently by two agents. Thus, lifting the table on one side is an action, but also lifting the table on both sides. In this case, there is no need to add concurrency conditions, and we need only change MA-STRIPS to allow for associating a set of agents with an action – the agents responsible for its execution. We illustrate both options using the box-pushing example.

```
BB: push(agent,box,location)
pre: at(agent,location) & at(box,location);
concurrency: push(agent',box,location)
                          & agent'!= agent;
effect: NOT at(box,location).

Joint: push(agent1,agent2,box,location)
pre: at(agent1, location) & at(agent2,location)
                           & at(box,location);
effect: NOT at(box,location).
```

The advantage of the BB syntax is that it is explicit about both positive and negative interactions, and allows us to more efficiently parallelize non-interacting actions. If we use the joint-actions approach, which essentially describes only positive interactions, we shall have to assume that all actions have negative interactions, and must be executed sequentially, or provide some other mechanism for identifying negative interactions.

Once we allow interacting actions, we examine if they impact the definitions of public and private variables and actions. With BB's syntax, actions remains associated with a single agent, and so the definition is not affected. With the joint-action syntax, if we view this as two copies of the same action belonging to two agents, all its preconditions and effects to be public. This appear too liberal (privacy-wise) because, eventually, a joint action is executed by two agents that simultaneously send an appropriate control signal.[1] At

[1] Extending these ideas to more than two actions is simple.

planning time, it is enough for each agent to know that the other agent will be executing its part of the action at the right time, and that might depend on the value of private preconditions known to that agent, but not to the other agent. Thus it is possible that some preconditions would be private to one agent, whereas others would be private to the other – as would be the case under the BB syntax. Consequently, a more conservative semantics stipulates (as in the current semantics):n a proposition is private to agent i if it is affected or required only by an action in which agent i *participates*. With this definition, we will have the same set of private variables as in the BB case, for natural formulation of actions and joint actions.

An interesting observation is that, in principle, a joint action can be private to the two agents. We discuss more refined notions of privacy later on. For the time being, we will consider all joint actions as public, and throughout the rest of this paper, we will stick with the joint-action approach, rather than BB, as it is simpler. We will also assume that the set of agents with which a joint action is tagged is ordered, which will help us to prevent multiple applications of the same joint actions by different agents at the same state.

MA Forward Search with Interacting Actions

We now review and extend the MAFS algorithm of Nissim and Brafman. See (Nissim and Brafman 2012; 2013a) for the full details. MAFS is a distributed version of best-first search – a similar version of A^* exists, and can be extended similarly. This algorithm maintains a separate search space for each agent. Each agent maintains an *open list* of states that are candidates for expansion and a *closed list* of already expanded states. It expands the state with the minimal f value in its open list. When an agent expands state s, *it uses its own operators only.* This means two agents expanding the same state will generate *different* successor states.

Since no agent expands all relevant search nodes, messages must be sent between agents, informing one agent of open search nodes relevant to it expanded by another agent. Agent φ_i characterizes state s as relevant to agent φ_j if φ_j has a public operator whose public preconditions (the preconditions φ_i is aware of) hold in s. In that case, Agent φ_i will send s to Agent φ_j.

The messages sent between agents contain the full state s, i.e., including both public and private variable values, as well as the cost of the best plan from the initial state to s found so far, and the sending agent's heuristic estimate of s. The private part of each state is encrypted by the relevant agent. Since this private part is not affected by other agents' actions, they can simply maintain this encrypted part as is.

When agent φ receives a state via a message, it checks whether this state exists in its open or closed lists. If it does not appear in these lists, it is inserted into the open list. If a copy of this state with higher g value exists, its g value is updated, and if it is in the closed list, it is reopened. Otherwise, it is discarded. Whenever a received state is (re)inserted into the open list, the agent computes its local h value for this state, and then can choose between/combine the value it has calculated and the h value in the received message. If both heuristics are known to be admissible, for example, the agent could choose the maximal of the two estimates.

Once an agent expands a solution state s, it sends s to all agents and awaits their confirmation. For simplicity, and in order to avoid deadlock, once an agent either broadcasts or confirms a solution, it is not allowed to create a new solution. If a solution is found by more than one agent, the one with lower cost is chosen, and ties are broken by choosing the solution of the agent having the lower ID. When the solution is confirmed by all agents, the agent initiates the trace-back of the solution plan. This is also a distributed process, which involves all agents that perform some action in the optimal plan. When the trace-back phase is done, a terminating message is broadcasted and the solution is outputted.

MAFS With Interacting Actions

Algorithms 1-3 depict the MAFS algorithm for agent φ_i. This algorithm supports interacting actions with minor modifications. The pseudo-code of the algorithms supporting joint-actions is provided below. Unfortunately, although the changes from the original algorithm (see (Nissim and Brafman 2013a)) are conceptually simple, the pseudo-code becomes much more complicated. Below we explain the basic enhancements required to the original algorithm.

Algorithm 1 MAFS for agent φ_i

1: **while** did not receive **true** from a solution verification procedure **do**
2: **for all** messages m in message queue **do**
3: **process-message(**m**)**
4: $(s, a, joint) \leftarrow$ **extract-min**(open list)
5: **expand(**$(s, a, joint)$**)**

Algorithm 2
process-message($m = \langle s, a, joint, g_{\varphi_j}(s), h_{\varphi_j}(s)\rangle$)

1: **if** joint **then**
2: add $(s, a, joint, g_{\varphi_j}(s), h_{\varphi_j}(s))$ to open list
3: **else if** s is not in open or closed list **or** $g_{\varphi_i}(s) > g_{\varphi_j}(s)$ **then**
4: calculate $h_{\varphi_i}(s)$
5: $g_{\varphi_i}(s) \leftarrow g_{\varphi_j}(s)$
6: $h_{\varphi_i}(s) \leftarrow max(h_{\varphi_i}(s), h_{\varphi_j}(s))$
7: add $(s, a, joint, g_{\varphi_i}(s), h_{\varphi_i}(s))$ to open list

Imagine that a is an interacting action involving agents i (the first agent) and j (the second agent). If at state s, the public preconditions of a and those private to i are satisfied, agent i will apply a and changing *only* the variables in s that are private to it; yielding a new state s', which is then sent to agent j, only. The message contains a and a special flag indicating it is the result of a joint action. Agent j examines state s'. If the public preconditions of action a and the preconditions private to j are satisfied in s', agent j will apply a, projected to its public variables and variables private to j, to s'. It will now continue with the regular algorithm. If, on the other hand, the preconditions of a that are private to j do not hold in s', (and therefore, in s), j will discard this state, sending it to no one. These changes enforce the semantics

Algorithm 3 expand($(s, a, joint, g, h)$)

if joint **then**
  if a's preconditions private to φ_i hold in s **then**
    $s' =$ apply a, projected to φ_i, to s
    Compute $h_{\varphi_i}(s')$
    for all agents $\varphi_j \in \Phi$ **do**
      if the last action leading to s was public **and** φ_j has a public action for which all public preconditions hold in s **then**
        send $(s', a, false, g, h_{\varphi_i}(s'))$ to φ_j
if $\neg$ joint **then**
  if s is a goal state **then**
    broadcast s to all agents
    initiate verification of s as a solution
    return
  else
    for all agents $\varphi_j \in \Phi$ **do**
      if the last action leading to s was public **and** φ_j has a public action for which all public preconditions hold in s **then**
        send s to φ_j
    for all non-joint actions a of φ_i applicable at s **do**
      for all successors s' generated by a **do**
        update $g_{\varphi_i}(s')$ and calculate $h_{\varphi_i}(s')$
        if s' is not in closed list **or** $f_{\varphi_i}(s')$ is now smaller than it was when s' was moved to closed list **then**
          move $(s', a, false, g_{\varphi_i}(s'), h_{\varphi_i}(s'))$ to open list
    for all joint actions a of φ_i applicable at s **do**
      for all successors s' generated by applying a projected to φ_i's private variables **do**
        update $g_{\varphi_i}(s')$ with a's cost
        send $m = (s', a, true, g_{\varphi_i}(s'), h_{\varphi_i}(s))$ to other agent taking part in a

of a joint action – a new state is formed only if the joint action is truly applicable. They also preserve privacy, as only the agent alters its private state and checks the validity of private preconditions, which are known to it alone.

The situation is slightly more complicated given conditional effects with private effect conditions and private conditional effects. In that case, the above algorithm need to be further modified, as the change in the public state and private state depends on whether the effect condition holds, and the agent i cannot know this ahead of time if the condition involves private variables of agent j. Instead, we require two, more complex messages. In the first message, agent i sends state s and marks the conditional effects whose effect condition are satisfied in s, as far as it knows (that is, excluding conditions private to j). Now, if a is applicable, agent j can apply a, projected to its private variables and the public variables. It sends the updated state s' to agent i, noting which of the conditional effects are applicable. Agent i now updates s' with the relevant changes to its private state.

Experimental Results

We implemented the MAFS algorithm with joint action on the Nissim's MAFD framework (Nissim and Brafman 2013b). Experiments were run on on an AMD Phenom 9550 2.2GHZ processor with multiple cores. Each agent executes on a single core. MAFD is an extension of the FD planner to MA-STRIPS. We used the optimal MAD-A* algorithm with the merge and shrink heuristic (Helmert, Haslum, and Hoffmann 2008). As we are not aware of another planner that can handle joint actions, we're simply providing running times and node expansion to get an idea of scalability.

Experiments were run on a box pushing domain and the results appear in Table 1. The domain consists of a grid of varying sizes, with a number of boxes that need to be pushed by the agents from the initial position to their goal positions. Small boxes can be pushed by a single agent, medium boxes require two agents, and large boxes require three agents. We show both the maximal time per agent and the combined CPU time of all agents (in seconds). As can be seen, increasing the number of boxes that require joint-actions to handle, seems to lead to substantial increase in the number of nodes expanded and generated, and a corresponding increase in running time. We intend to run additional experiments that attempt to better isolate the impact of joint vs. non-joint actions on planning effort.

Open Issues for MA-STRIPS

Compilation. Is it possible to compile a domain with joint-actions into a domain without joint actions so that existing MA planners without compromising privacy?

Similar Actions and Task Allocations. When different agents can achieve the same proposition, task allocation – who should achieve this proposition – becomes an issue. A special case is when agents have identical, or almost identical actions. Although MA-STRIPS associates one agent with each action, the action set could contain multiple identical actions with different ids, each associated with a different agent, or actions with identical public parts, but different private parts. A simple, intuitive example is that of two trucks that can serve similar locations. Both can pickup or drop a package in these same location, but their private preconditions are different, each pertaining to the location of the respective truck.

In that case, it may beneficial to artificially divide locations between the two agents, so that only one truck can service each location. This results in many more private variables and actions, allowing for decoupled planning by the agents. Indeed, if we consider the complexity of analysis of Brafman and Domshlak (Brafman and Domshlak 2013), as the set of public actions becomes smaller, the problem of coordinating public aspects of plans becomes easier. The practical effect of having fewer public actions and more decoupled actions sets can be seen in Nissim et al. (Nissim, Apsel, and Brafman 2012).

Of course, by restricting the behavior of agents, we are potentially removing possible solutions, perhaps even all solutions, and certainly optimal ones. While we conjecture that the problem of finding such restrictions – which correspond

problem	agents	grid size	Boxes (s,m,l)	Expanded	Generated	Messages	Max Time	Total Time
Box1	2	1x3	3,0,0	90	167	29	0.04	0.08
Box2	2	1x3	2,1,0	134	344	126	0.08	0.016
Box3	2	1x6	2,2,0	114	311	105	0.03	0.06
Box4	2	4x4	3,1,0	71	291	66	0.04	0.08
Box5	2	6x6	0,2,0	97	353	97	0.13	0.26
Box6	2	6x6	0,4,0	31409	137797	31403	1.72	3.39
Box7	3	6x6	3,1,1	750901	3794933	1501760	29.92	88.65

Table 1: Performance of Joint-Actions Enhanced MAFS on Box Pushing Domain

in some sense to task allocation – is as hard as solving the planning problem, we believe that the simple MA-STRIPS framework provides an opportunity to study this question both theoretically and practically.

Private Actions. Work on MA-STRIPS divides actions and variables to two classes: private and public. As noted, the more actions are private, the more we can decouple the problem. In task allocation, discussed above, we artificially increases the set of private actions. We may be able to decouple farther by introducing finer grained distinctions and exploiting them in planning algorithms. For example, some actions are public to a limited set of agents. That is, if we were to combine these agents to a single agent, the action would become private. A hierarchical algorithms might be able to exploit this structure by building a hierarchy of partitions into agents sets.

Another interesting option is to differentiate among agents that require a variable between those that can and cannot change its value. Thus, some public variables may be "read-only public" because only one agent can change them, although others may need them. This is, essentially, the distinction between preconditions and prevail conditions made by the SAS+ formalism (Bäckström and Klein 1991). Thus, some variables may be private to an agent, some may be public, and some may be changeable only by that agent, making them somewhere between private and public. Whether this distinction can be useful either in practice or for a more refined theoretical analysis is an open question.

Summary

We believe there is much merit in understanding fundamental issues by studying simple models of multi-agent planning. In this spirit, we tried to minimally extend MA-STRIPS and, in particular, the MAFS schema, to address the problem of interacting actions and implemented our algorithm within the MAFD platform developed by Raz Nissim. We also discusses some other potential issues that can be addressed within the MA-STRIPS model: task allocation and refined ideas of private/public actions and variables.

Acknowledgements: We thank the reviewers for their useful comments. The authors are supported in part by by ISF Grant 933/13. Brafman and Shimony are supported in part by the Lynn and William Frankel Center for Computer Science.

References

Bäckström, C., and Klein, I. 1991. Planning in polynomial time: The SAS-PUBS class. *Computational Intelligence* 7(3):181–197.

Boutilier, C., and Brafman, R. I. 2001. Planing with concurrent interacting actions. *Journal of AI Research* 14:105–136.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, 28–35.

Brafman, R. I., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artif. Intell.* 198:52–71.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning games. In *IJCAI*, 73–78.

Cox, J. S., and Durfee, E. H. 2005. An efficient algorithm for multiagent plan coordination. In *AAMAS*, 828–835. ACM.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2008. Explicit-state abstraction: A new method for generating heuristic functions. In *AAAI*, 1547–1550.

Kovacs, D. L. 2012. A multi-agent extension of pddl3.1. In *3rd ICAPS Workshop on the International Planning Competition*.

Nissim, R., and Brafman, R. I. 2012. Multi-agent a* for parallel and distributed systems. In *AAMAS*, 1265–1266.

Nissim, R., and Brafman, R. I. 2013a. Distributed heuristic forward search for multi-agent systems. *CoRR* abs/1306.5858.

Nissim, R., and Brafman, R. I. 2013b. Distributed heuristic forward search for multi-agent systems. *CoRR* abs/1306.5858.

Nissim, R.; Apsel, U.; and Brafman, R. I. 2012. Tunneling and decomposition-based state reduction for optimal planning. In *ECAI*, 624–629.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *AAMAS*, 1323–1330.

Steenhuisen, J. R.; Witteveen, C.; ter Mors, A.; and Valk, J. 2006. Framework and complexity results for coordinating non-cooperative planning agents. In *MATES*, 98–109.

Szer, D.; Charpillet, F.; and Zilberstein, S. 2005. Maa*: A heuristic search algorithm for solving decentralized pomdps. In *UAI*, 576–590.

ter Mors, A., and Witteveen, C. 2005. Coordinating self interested autonomous planning agents. In *BNAIC*, 383–384.

Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *ECAI*, 762–767.

Multiagent Planning by Iterative Negotiation over Distributed Planning Graphs

Jan Tožička[1], **Jan Jakubův**[1], **Karel Durkota**[1], **Antonín Komenda**[2]
[1] Agent Technology Center, CTU in Prague
Prague, Czech Republic
[2] Technion - Israel Institute of Technolog
Haifa, Israel

Abstract

Multiagent planning for cooperative agents in deterministic environments intertwines *synthesis* and *coordination* of the local plans of involved agents. Both of these processes require an underlying structure to describe synchronization of the plans. A *distributed planning graph* can act as such a structure, benefiting by its compact representation and efficient building. In this paper, we propose a general negotiation scheme for multiagent planning based on planning graphs. The scheme is designed as independent on a particular local plan synthesis approach.

To demonstrate the proposed principle, we have implemented the negotiation scheme as an algorithm with a concrete technique for the local plan synthesis based on compilation of the local planning problems to SAT problems. Results of the negotiation further shape the SAT problems so that agents coordinate their plans and avoid possible conflicts in an iterative manner. The paper is concluded by showing a set of experiments which demonstrate a trade-off between planning efficiency (by means of time and communication) and increasing amount of public information in the planning problem.

Introduction

Intelligent agents embodied in an environment have to be able not only to selfishly push the world towards their own goals but also to cooperate on common goals with their neighbors and solve mutual conflicts if their plans interfere. *Multiagent planning*, an umbrella term for such behavior, deals with challenges both on (i) the synthesis of actions into individual agents' plans and on (ii) the coordination of the agents' plans in a shared environment. A generally usable approach to multiagent planning has to be able to deal with a wide range of application domains without any fixed domain-specific knowledge.

In such cases of domain-independent planning, the input to the planning process does not contain only the initial and goal conditions on the environment, but also description of the problem domain. Such approaches allow the agents to prepare plans flexibly according to their knowledge of the environment mechanics. From the practical perspective, domain-independent techniques can be reused in various circumstances, where the agents are required to plan.

In one of the most cited works on multiagent planning (Durfee 1999), Durfee describes basics of possible coordination schemes for planning agents. From the taxonomy presented there, our approach falls into *a distributed planning of distributed plans* which do not assume either the planning process or the resulting plan to be centralized. There is a large amount of work dealing with another facets of the coordination part of the problem, e.g. GPGP (Decker and Lesser 1992), or TALPlanner (Doherty and Kvarnström 2001). In 2008, Brafman and Domshlak proposed multiagent planning in (Brafman and Domshlak 2008) which targeted deterministic environments and was based on an extension of the classical planning model STRIPS (Fikes and Nilsson 1971). The results of the paper showed that deterministic domain-independent multiagent planning is not exponentially dependent on the number of agents in the computational sense.

The approach we propose in this paper can be seen as a merge and extension of two previous approaches. The first one is from Zhang et al. presented in (Zhang, Nguyen, and Kowalczyk 2007). The idea behind it is based on distribution of a well-known structure—*a planning graph*—and compilation of the planning problem into a DisCSP problem. The other approach authored by Pellier in (Pellier 2010) is also based on distributed planning graphs, however for the local plan extraction each agent uses a centralized CSP solver and the coordination of their plans is done by a backtracking approach resembling prioritized planning.

Our contribution in this work is threefold. Firstly, we have generalized the predetermined coordination part (done by Zhang at al. as DisCSP and Pellier as prioritized planning) by a decentralized approach based on novel negotiation scheme. This negotiation scheme extends our scheme published in (Tožička et al. 2014) by handling new types of other agent responses. Secondly, we propose a way how to supersede the Pellier's CSP-based extraction of local plans by compilation to SAT problems. And finally, we have designed and implemented an extension of the planning graph structure by state-of-the-art planning modeling approach SAS+ (Huang, Chen, and Zhang 2012). We also experimentally show a trade-off between planning efficiency (by means of computation time and communication) and increasing amount of public information in the planning problem.

Planning Model

We consider a number of *cooperative* and *coordinated* agents featuring distinct sets of capabilities (actions) which concurrently plan and execute their local plans in order to achieve a joint goal. The environment wherein the agents act is *classical* with *deterministic* actions. The following formal preliminaries compactly restate the MA-STRIPS problem (Brafman and Domshlak 2008) required for the following sections.

This model, together with proofs of lemmas and theorems, has been already published in (Tožička et al. 2014). Nevertheless, we consider it necessary to repeat basic definitions and lemmas to make paper standalone.

Planning Problem

An MA-STRIPS planning problem Π is defined as a quadruple $\Pi = \langle P, \mathcal{A}, I, G\rangle$, where P is a set of propositions, $\mathcal{A}$ is a set of *agents* $\alpha_1, \ldots, \alpha_{|\mathcal{A}|}$, I is an initial state and G is a set of goals.

An *action* an agent can perform is a triple $a = \langle a_{\mathsf{pre}}, a_{\mathsf{add}}, a_{\mathsf{del}}\rangle$ of subsets of P, where a_{pre} is the set of preconditions, a_{add} is the set of add effects, and a_{del} is the set of delete effects. We define functions $\mathsf{pre}(a)$, $\mathsf{add}(a)$, and $\mathsf{del}(a)$ such that for any action a it holds $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle$. Moreover let $\mathsf{eff}(a) = \mathsf{add}(a) \cup \mathsf{del}(a)$.

We identify an *agent* with its capabilities, that is, an agent $\alpha = \{a_1, \ldots, a_n\}$ is characterized by a finite repertoire of actions it can preform in the environment. Let A_Π denote the set of all actions in a problem Π, that is, $A_\Pi = \bigcup_{\alpha\in\mathcal{A}} \alpha$. A *state* $s = \{p_1, \ldots, p_m\} \subseteq P$ is a finite set of facts and we say that p_i's hold in s.

Public and Internal Actions

MA-STRIPS problems distinguish between the *public* and *internal* facts and actions. Let $\mathsf{facts}(a) = \mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$ and similarly $\mathsf{facts}(\alpha) = \bigcup_{a\in\alpha} \mathsf{facts}(a)$. An α-internal and public subset of all facts P, denoted $P^{\alpha\text{-int}}$ and P^{pub} respectively, are subsets of P such that the following hold.

$$\begin{aligned} P^{\text{pub}} &\supseteq \bigcup_{\alpha\in\mathcal{A}} \left(\mathsf{facts}(\alpha) \cap \bigcup_{\beta\in\mathcal{A}\setminus\{\alpha\}} \mathsf{facts}(\beta)\right) \\ P^{\alpha\text{-int}} &= \mathsf{facts}(\alpha) \setminus P^{\text{pub}} \end{aligned}$$

Set P^{pub} contains all the facts that are used in actions of at least two different agents. The set can possibly contain also other facts, that is, some facts mentioned in actions of one agent only. This definition of public facts differs from other definitions in literature (Brafman and Domshlak 2008) where P^{pub} is defined using equality instead of superset ($\supseteq$). Our definition, however, allows us to experiment with extensions of the set of public facts and this is discussed below in experiment section. We suppose that P^{pub} is an arbitrary but fixed set which satisfies the above condition. Set $P^{\alpha\text{-int}}$ of α-internal facts contains facts mentioned only in the actions of agent α, but possibly not all of them.

The set P^α of facts *relevant* for a single agent α is defined as $P^\alpha = P^{\alpha\text{-int}} \cup P^{\text{pub}}$. The *projection* a^S of an action a to a set of facts S is a restriction of a containing only facts from S, that is, $a^S = \langle \mathsf{pre}(a) \cap S, \mathsf{add}(a) \cap S, \mathsf{del}(a) \cap S\rangle$. The projection a^α of action a to agent α is defined as $a^{(P^\alpha)}$ and the public projection a^{pub} of action a is defined as $a^{(P^{\text{pub}})}$.

The set α^{pub} of *public actions* of agent α is defined as $\alpha^{\text{pub}} = \{a \mid a \in \alpha, \mathsf{eff}(a^{\text{pub}}) \neq \emptyset\}$ and the set α^{int} of *internal actions* of agent α as $\alpha^{\text{int}} = \alpha \setminus \alpha^{\text{pub}}$. The set A^{pub}_Π of *all public actions* of problem Π is defined as $A^{\text{pub}}_\Pi = \bigcup_{\alpha\in\mathcal{A}} \alpha^{\text{pub}}$. A^α_Π the set of all actions known by agent α is then $A^\alpha_\Pi = \alpha^{\text{int}} \cup \{a^\alpha | a \in A^{\text{pub}}_\Pi\}$.

In the rest of this paper we consider only problems Π where all the propositions from the goal state G are public, that is, $G \subseteq P^{\text{pub}}$ which is common in literature (Nissim and Brafman 2012)[1]. Moreover we suppose that two different agents do not execute the same action, that is, we suppose that the sets α_i's are pairwise disjoint (Brafman and Domshlak 2008).

Plans, Solutions, and Projections

The projection Π^α of a problem Π to agent α is a classical STRIPS problem defined as follows.

$$\Pi^\alpha = \langle P^\alpha, A^\alpha_\Pi, I \cap P^\alpha, G\rangle$$

Given an MA-STRIPS problem Π, a *plan* $\pi = \langle a_1, \ldots, a_k\rangle$ is a sequence of actions from A_Π. A plan π defines an order in which actions are to be executed by their unique owner agents. It is supposed that independent actions can be executed in parallel. A plan π is called a *solution* of Π if a sequential execution of the actions from π by their respective owners transforms the initial state I to a subset of the goal G. Let $\mathsf{sol}(\Pi)$ denote the set of all solutions of problem Π. We use $\pi[i]$ to denote a_i, that is, the action from the plan at position i. Moreover $\pi[i \ldots j]$ where $i \leq j$ denotes the plan subsequence $\langle a_i, \ldots, a_j\rangle$.

The *projection* π^{S} of a plan $\pi = \langle a_1, \ldots, a_k\rangle$ is computed from π by projecting each action a_i to S and by subsequent removal of empty projections. Formally we define

$$\pi \overset{S}{\longmapsto} \pi^{\text{S}} \overset{\text{def}}{\iff} \pi^{\text{S}} = \langle a^{\text{S}}_1, \ldots, a^{\text{S}}_k\rangle \,|^{A^{\text{S}}_\Pi},$$

where the restriction $|^{A^{\text{S}}_\Pi}$ creates a subsequence containing only actions of A^{S}_Π. Note that different plans can have the same projection. The *public plan projection* π^{pub} is defined as $\pi^{(A^{\text{pub}}_\Pi)}$ and the plan projection π^α to agent α is defined as $\pi^{(P^\alpha)}$. A plan is called *public* w.r.t. Π if $a_i \in A^{\text{pub}}_\Pi$ for all i.

Extensible Plans

The following defines *(α)-internally extensible* plans which are plans that can be transformed to a solution by inserting only internal actions into it.

Definition 1. *Let Π be* MA-STRIPS *problem and let π be a plan public w.r.t. Π. We say that the public plan π is* α-internally extensible *if*

$$\exists \pi' \in \mathsf{sol}(\Pi^\alpha) : \pi' \overset{\text{pub}}{\longmapsto} \pi$$

[1] This condition can be weakened, but we stick to it as it simplifies this paper.

We say that the public plan π is internally extensible *if*

$$\exists \pi' \in \mathsf{sol}(\Pi) : \pi' \overset{\text{pub}}{\longmapsto} \pi$$

The following lemma states that a solution of problem Π can be obtained composing partial α-internally extensible plans of all the involved agents.

Lemma 1. *Let Π be* MA-STRIPS *problem and let π be a plan public w.r.t. Π. A plan π is* α-internally extensible *for every agent α that owns some action from π if and only if π is* internally extensible*.*

Similarly to the Definition 1 the following defines a *publicly extensible* plan which can be transformed to a solution by inserting both public and internal actions into it.

Definition 2. *Let Π be given. We say that a plan π is* publicly extensible *if there exists a plan π' which is* internally extensible *and such that π is a subsequence of π'.*

Plan Domains

The following defines a plan *domain* $\mathcal{D}$ which is a key structure used in our algorithms. A domain $\mathcal{D}$ is a set of plans with operations defined as follows.

$$\begin{array}{lcl} \mathcal{D} \ominus \pi & = & \mathcal{D} \setminus \{\pi\} \\ \mathcal{D} \oplus \langle a, t\rangle & = & \{\pi \in \mathcal{D} | \pi[t] = a\} \\ \mathcal{D} \ominus \langle a, t\rangle & = & \mathcal{D} \setminus (\mathcal{D} \oplus \langle a, t\rangle) \end{array}$$

Moreover let $\mathcal{D}^{l_{\mathsf{max}}}$ denote the set of all plans of length l_{max}.

In our algorithms presented in the following sections we suppose that we have a classical planner which computes a solution of a given classical planning problem which is in a given domain $\mathcal{D}$, that is, that we have an effective procedure that selects a solution from a given domain. We work with plan domains defined as sets of plans to abstract from a concrete implementation and to simplify presentation of the algorithms in the following sections. A plan domain $\mathcal{D}$ can be seen as an abstract data structure which supports the above three operations and whose semantics is defined using aforementioned sets of plans. Our effective implementation of plan domains uses planning graphs and a SAT solver. We encode the search for a plan in a planning graph as a SAT instance and operations on domain $\mathcal{D}$ then add additional conditions to the SAT instance so that the search is restricted to $\mathcal{D}$. The implementation is further described below.

Operation $\mathcal{D} \ominus \pi$ simply removes π from the domain. Operation $\mathcal{D} \oplus \langle a, t\rangle$ restricts the domain so that it contains only those plans which contain action a at position t. Finally, operation $\mathcal{D} \ominus \langle a, t\rangle$ does exactly the opposite, that is, it restricts the domain so that it contains only those plans which do not contain action a at position t.

Confirmation Scheme

In this section we present a multiagent planning algorithm which effectively iterates over all plans in order to find internally extensible solution. This *confirmation* algorithm can also be seen as a skeleton which is further elaborated in next section. The confirmation algorithm provides a sound and complete multiagent planning algorithm (see Theorem 2).

Algorithm 1: Multiagent planning algorithm with iterative deepening.

input: multiagent planning problem Π
output: a solution π of Π when solution exists
Function `MultiPlanIterative`(Π) **is**
  $l_{\mathsf{max}} \leftarrow 1$
  loop
    $\pi \leftarrow$ `MultiPlan`(Π, l_{max})
    if $\pi \neq \emptyset$ **then**
      return π
    end
    $l_{\mathsf{max}} \leftarrow l_{\mathsf{max}} + 1$
  end
end

Procedure `MultiplanIterative` from Algorithm 1 is the main entry point of our algorithms, both in this and the following sections. This procedure is initially executed by one of the agents called *initiator*. It takes a problem Π as the only argument and it iteratively calls procedure `MultiPlan`(Π, l_{max}) to find a solution of Π of length l_{max}, increasing l_{max} by one on a failure. In this way we ensure completeness of our algorithm because we enumerate the infinite set of all plans in a way that does not miss any solution. To simplify the presentation, we restrict our research only to those problems Π which actually have a solution.

Algorithm 2: `MultiPlan`(Π, l_{max}) in the confirmation scheme. Method `SinglePlan`($\Pi, \mathcal{D}$) returns a plan from domain $\mathcal{D}$ solving problem Π or $\emptyset$ if there is no such plan. Constructor `PlanDomain` constructs a plan domain with a given semantics. Method `AskAllAgents`(π^{pub}) asks all agents mentioned in the plan whether they consider the *public projection* of this plan to be *internally extensible* and returns `OK` if all agents reply `YES`.

input: problem Π and a maximum plan length l_{max}
output: a solution π of Π when solution exists
Function `MultiPlan`(Π, l_{max}) **is**
  $\mathcal{D} \leftarrow$ **new** `PlanDomain`($\{\pi : |\pi| = l_{\mathsf{max}}\}$)
  loop
    $\pi \leftarrow$ `SinglePlan`($\Pi, \mathcal{D}$)
    if $\pi = \emptyset$ **then**
      return $\emptyset$
    end
    $reply \leftarrow$ `AskAllAgents`(π^{pub})
    if $reply =$ `OK` **then**
      return π
    end
    $\mathcal{D} \leftarrow \mathcal{D} \ominus \pi$
  end
end

Algorithm 2 presents implementation of `MultiPlan` in the confirmation algorithm. Operator `PlanDomain` con-

structs a planning domain with semantics described by its argument. We suppose that $\texttt{SinglePlan}(\Pi, \mathcal{D})$ implements a sound and complete classical planner which returns a solution of Π^α within a given domain $\mathcal{D}$ where α correspond to the agent executing the task. Moreover we suppose that SinglePlan always terminates and that it returns $\emptyset$ when there is no solution. Our effective implementation of SinglePlan is described in local plan extraction section.

Initially, we create a domain that contains all the plans of length l_{max}. Then we invoke SinglePlan to obtain a solution of Π^α denoted as π. Afterwards, we ask all involved agents whether or not the public projection π^{pub} is internally extensible. To answer this question, each agent invokes SinglePlan for a problem considering only actions from π^{pub} together with its internal actions while using a plan domain to describe possible partial solutions. When the answers from all of the agents are affirmative then π is returned as a result. Otherwise π is excluded from domain $\mathcal{D}$ and SinglePlan is called to compute a different solution.

The following states that the plan returned by the confirmation algorithm is internally extensible to a solution of Π (*soundness*), and that the algorithm finds internally extensible solution when there is one (*completeness*). It is easy to construct a solution of Π given an internally extensible plan.

Theorem 2. *Algorithm* MultiplanIterative *(Alg. 1) with confirmation procedure* MultiPlan *(Alg. 2) is sound and complete.*

Iterative Negotiation Scheme

A drawback of the confirmation scheme from the previous section is that it requires an initiator agent to find a plan which is internally extensible to a problem solution. It means that the other agents, called *participants*, can insert only their internal actions into the plan and this can be too limiting. Our iterative negotiation algorithm from this section tries to overcome this drawback by attempting to correct a publicly extensible plan to a solution. Hence we distinguish the following cases depending on a result π returned by SinglePlan.

CASE-I **– π is an *internally extensible* plan** – all participants can extend the plan adding internal actions only.

CASE-II **– π is a *publicly extensible* plan** – all participants can extend the plan but some can require another public action to be performed prior to their action.

CASE-III **– Otherwise** – negotiation fails, the initiator restricts the domain $\mathcal{D}$ and replans.

The confirmation algorithm handles only situations described in CASE-I. Plans of CASE-II are handled as CASE-III, that is, the search for an internally extensible plan continues in a restricted domain. The following subsections describe improved handling of CASE-I and CASE-II in the iterative negotiation algorithm.

Handling of Internally Extensible Plans

Handling of CASE-I in the iterative negotiation algorithm is presented in Algorithm 3. One of the agents, called *initiator*, starts the negotiation. Other agents are called *partici-*

Algorithm 3: $\texttt{MultiPlan}(\Pi, l_{\mathsf{max}})$ in iterative negotiation scheme: Systematic search through all possible plans with backtracking (confirming actions from the beginning of the plan). Procedure $\texttt{AskAgent}(\alpha, \pi)$ queries agent α how it rates the plan π. It returns CASE-I if it is prefix of some *internally extensible* plan, otherwise returns CASE-II if it is prefix of some *publicly extensible* plan, otherwise it returns CASE-III.

input: problem Π and a maximum plan length l_{max}
output: a solution π of Π when it exists, $\emptyset$ otherwise
Function $\texttt{MultiPlan}(\Pi, l_{\mathsf{max}})$ **is**
  $\mathcal{D}^{l_{\mathsf{max}}} \leftarrow$ **new** $\texttt{PlanDomain}(\{\pi : |\pi| = l_{\mathsf{max}}\})$
  $\pi \leftarrow \texttt{SinglePlan}(\Pi, \mathcal{D}^{l_{\mathsf{max}}})$
  $\pi^{\checkmark} \leftarrow \texttt{CorrectPlan}(\pi, 1, \Pi, \mathcal{D}^{l_{\mathsf{max}}})$
  return $\pi^{\checkmark}$
end

input: plan π, index of first action that can be changed l, problem Π and the domain $\mathcal{D}$
output: a solution π of Π when it exists, $\emptyset$ otherwise
Function $\texttt{CorrectPlan}(\pi, l, \Pi, \mathcal{D})$ **is**
  repeat
    $\alpha \leftarrow \texttt{OwnerOf}(\pi[l])$
    reply $\leftarrow \texttt{AskAgent}(\alpha, (\pi[1 \dots l])^{\mathrm{pub}})$
    switch *reply* **do**
      case CASE-I
        if $|\pi| = l$ **then**
          return π
        end
        $\pi^{\checkmark} \leftarrow$
        $\texttt{CorrectPlan}(\pi, l+1, \Pi, \mathcal{D} \oplus \langle \pi[l], l \rangle)$
        if $\pi^{\checkmark} \neq \emptyset$ **then**
          return $\pi^{\checkmark}$
        end
      end
      case CASE-II
        /* Do nothing for now, will be handled by Algorithm 4. */
      end
      otherwise (CASE-III)
        /* Do nothing. Subplan $\pi[1 \dots l]$ is not prefix of any solution. */
      end
    endsw
    $\mathcal{D} \leftarrow \mathcal{D} \ominus \langle \pi[l], l \rangle$
    $\pi \leftarrow \texttt{SinglePlan}(\Pi, \mathcal{D})$
  until $\pi = \emptyset$
  return $\emptyset$
end

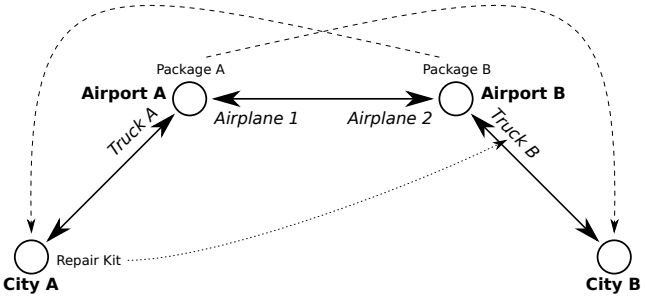


Figure 1: Logistics problem. Two packages need to be transported as shown by dashed arrows. Each of four vehicles can transport objects only between a pair of neighboring locations. In one analyzed variant, the `TruckB` is broken and it thus requires be fixed using the `RepairKit` before it can move anywhere.

pants. The initiator selects its local solution from the initial domain $\mathcal{D}^{l_{\max}}$ a tries to correct it to an internally extensible solution of a given problem. Procedure `CorrectPlan` iterates over the actions from π a tries to confirm the actions one by one, by asking the action owner to confirm action position. If the answer of `AskAgent` is `CASE-I` then it continues to query the next action. Otherwise the initiator remembers that this action can not be performed at specified position (under assumption that previously confirmed actions precede) and tries to find a different plan where this action is not required, while the previous actions in the plan remain the same. The following example illustrates Algorithm 3.

Example 1. *Let us demonstrate our algorithms on a simple logistics problem illustrated by Figure 1. They are two packages located at two airports. The goal is to transport them to distant cities. In order to achieve this goal, it is necessary to transport each package to the second airport by a plane and then to load it onto the truck and move it to the target city.*

Let's suppose that all the facts describing the location of the packages are public and all other facts are internal. Let's also suppose that the agent `AirplaneA` *decides to solve this task and starts the planning process. In the first run of planning process, it creates a plan that seems to solve the problem. An example of such a plan follows:*

```
1. load(AirplaneB, PackageB, AirportB)
2. unload(AirplaneB, PackageB, AirportA)
3. load(AirplaneA, PackageA, AirportA)
4. fly(AirplaneA, AirportA, AirportB)
5. unload(AirplaneA, PackageA, AirportB)
6. load(TruckA, PackageB, AirportA)
7. load(TruckB, PackageA, AirportB)
8. unload(TruckA, PackageB, CityA)
9. unload(TruckB, PackageA, CityB)
```

Now, the `AirplaneA` *verifies that all the agents, that are required to perform some action in the plan, can really perform the requested action. Agent* `AirplaneA` *first asks* `AirplaneB` *to load* `PackageB` *at* `AirportB` *at time 1. This is directly possible and thus the* `AirplaneB` *replies with* `CASE-I`. `AirplaneA` *then queries* `AirplaneB` *with first two actions.* `AirplaneB` *cannot perform the* `unload` *action immediately after the* `load` *action, nevertheless it is required to insert only one internal action* `fly` *to create a valid plan that can be prefix of some solution. Therefore,* `AirplaneB` *replies with* `CASE-I` *again.*

Similarly, the negotiation continues action by action to the end of the plan and then agent `AirplaneA` *can confirm that the plan is* internally extensible.

Handling of Publicly Extensible Plans

It is a more complex problem to detect whether a plan is *publicly extensible* and then to convert it into a *internally extensible* plan. In this case, the initiator α creates a plan solving Π^{α} which misses some public actions required by some participant to allow him to cooperate on this plan. When the participant is queried with a plan containing such an action, it replies with `CASE-II` as demonstrated by Example 2. Then initiator asks him for a list of missing required public action. These actions are inserted into the original plan by the initiator and they have to be confirmed by owner agents. They can contain actions that cannot be performed – then the initiator asks the participant for some alternative plan that would allow him to perform required action. Additionally, the actions returned by the participant can also contain actions owned by another participant. These actions of another participant need again to be verified.

In the `CASE-II`, the plan extension is searched in depth-first manner and thus it can yield in infinite cycle in some cases. Therefore, there has to be some limitation to stop the deepening. It can be limited by the number of plan extensions '$\circ reqActs \circ$' or by the maximal length of π' (e.g. $|\pi'| \leq 2 \cdot l_{\max}$).

Algorithm 4: If the participant marks the plan as *publicly extensible* (`CASE-II`), the initiator asks him for missing public actions using method `askRequiredActions`(α). These actions are then inserted into the current plan π.

case `CASE-II`
  while $(reqActs \leftarrow$ `askRequiredActions`$(\alpha)) \neq \emptyset$ **do**
    $\pi' \leftarrow \pi[1 \ldots (l-1)] \circ reqActs \circ \pi[l \ldots]$
    $\mathcal{D}' \leftarrow \{\pi_0 \in$
    $\mathcal{D} |\ \pi_0[1 \ldots (l-1)] \circ reqActs \circ \pi_0[l \ldots]\}$
    $\pi^{\checkmark} \leftarrow$ `CorrectPlan`$(\pi', l, \Pi, \mathcal{D}')$
    if $\pi^{\checkmark} \neq \emptyset$ **then**
      return $\pi^{\checkmark}$
    end
  end
end

Example 2. *Let's extend the previous example by* `TruckA`*'s internal state that it has broken engine and thus it cannot perform action* `unload` *requiring internal action* `move` *unless it is fixed. In order to perform the* `move`

action it has to `fixEngine` *using the* `RepairKit`*. The* `RepairKit` *is placed at* `CityB` *and thus it has to be transported by* `TruckB` *and a plane to the* `AirportA` *(location where* `TruckA` *is placed). There is no reason why the initiator would plan to move the* `RepairKit`[2]

When broken truck `TruckA` *is asked to to fulfill action* `unload` *at time 8, it creates plan containing following actions (apart from the action specified by the request):*

```
 8. load(TruckB, RepairKit, CityB)
 9. unload(TruckB, RepairKit, AirportB)
10. load(AirplaneA, RepairKit, AirportB)
11. unload(AirplaneA, RepairKit, AirportA)
12. fixEngine(TruckA, RepairKit, AirportA)
13. move(TruckA, CityA)
14. unload(TruckA, PackageB, CityA)
```

First four actions are public actions that need to be performed by other agents before the `TruckA` *can fix its engine and move to destination where it will unload the package. Therefore, the reply is* `CASE-II` *with a subplan containing these four public actions. Initiator will insert this subplan into his plan and continues the negotiation.*

Obviously, this approach can change *publicly extensible* plan into an *internally extensible* plan only if it is possible to insert required public actions immediately before the action which requires them. In some domains, this does not have to be true, and the *publicly extensibly* plan can require some public action to be inserted before some other already planned action. Nevertheless, this problem does not reduce the completeness of proposed algorithm, because the required action will be planned later by the initiator once it tries all other possible plans having that part of the plan fixed (this situation is handled similarly by Algorithm 3).

From theory to practice

We have implemented the algorithms described in the previous sections taking advantage of several existing techniques and systems. A overall scheme of the architecture of our planner is sketched at Figure 2. An input problem Π described in PDDL is translated into SAS using *Translator* script which is a part of Fast Downward[3] system. Our *Multi-SAS* script then splits SAS representation of the problem Π into agents' projections Π^α using user provided selection of public facts P^{pub}. Each agent can then compute its local *planning graph* up to level l_{max} where l_{max} is always increased by one on failure. Each planning graph represents a set of plans including all solutions if any exists. We then encode the search for a local solution in a planning graph into a SAT instance and we use MiniSat[4] solver to find a solution to the problem Π^α.

[2]Actions moving with the `RepairKit` are part of the domain $\mathcal{D}^{l_{\mathsf{max}}}$ (for some l_{max}) but let's suppose that we have a planner that prefers NOOPs to moving some object which is not required by the goal. Nevertheless, initiator uses complete method and thus it will come to the solution soon or later.

[3]`http://www.fast-downward.org/`

[4]`http://minisat.se/`

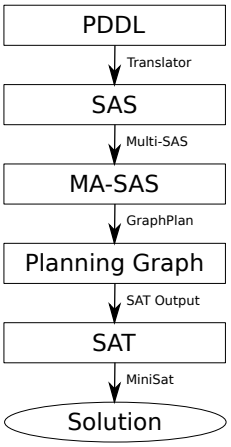


Figure 2: Architecture of the planner.

It is crucial to use suitable representation of plan domain $\mathcal{D}$ to allow effective implementation of all operators and function used in our algorithms under reasonable memory requirements. Direct listing of all the plans is obviously unfeasible. We have chosen to use *planning graphs* represented as SAT problems. In following sections we define multiagent planning graphs and describe (i) how to encode the search for a solution in a planning graph into a SAT instance and (ii) how the SAT instance is altered so that it encodes the search for a solution in a restricted domain. We conclude with a discussion of possible algorithm improvements.

Agent Planning Graphs

Agent planning graphs (APGs) stem from the classical planning graphs. Building distributed planning graphs was previously studied with focus on distribution of the Graphplan algorithm (Pellier 2010). Multiagent planning graphs were also studied recently in its relaxed form (Torreño, Onaindia, and Sapena 2012)[removed for review].

An APG is a directed, labeled and layered graph $\mathcal{G}^\alpha = (P \cup A, E)$ of one particular agent α for a local agent's planning task. As in a classical planning graph, the nodes of the graph represent propositions P and actions A. The arcs E represent linkup of propositions and actions.

In more detail, an *i*-th proposition layer and action layer will be denoted as P_i and A_i respectively. The layers alternate, so that $(P_0, A_0, P_1, A_1, \ldots, A_{n-1}, P_n)$ and all layers $P_i \subseteq P$ and all layers $A_i \subseteq A$. The first proposition layer P_0 contains nodes labeled by propositions of the agent's projection of the initial state: $P_0 = I \cap P^\alpha$.

Each action layer contains action nodes for all applicable actions of the agent α in a state represented by the previous fact layer and external projections of other agents' public actions reachable in the same layer $A_i = \{a | a \in A^\alpha_\Pi, \mathsf{pre}(a) \subseteq P_i\}$. In all successive fact layers, the nodes copy the previous fact layer according to the frame axiom and transforms the facts by actions in the previous action layer: $P_i = P_{i-1} \cup \{p | p \in \mathsf{add}(a), a \in A_{i-1}\}$.

Additionally, there is defined relation between pairs of ac-

tions and pairs of facts called *mutexes*. It is constructed during the creation of the planning graph using specific rules described in (Ghallab, Nau, and Traverso 2004). Whenever two actions or facts are in mutex, it means that they cannot be achieved simultaneously.

In our implementation the APG is built until all the facts of the goal are supported and there is no mutex between them. Then we add layers as described by Algorithm 1.

Local Plan Extraction

Initial domain $\mathcal{D}^{l_{\max}}$ for a *Planning graph* containing $l_{\max}$ layers in SAT representation contains a variable for each:

- action at each PG layer (including the *noop* actions) indicating whether the action is activated (i.e. part of the plan)
- fact at each PG layer indicating whether this atom is supported by some active action

Then it contains a formula for each:

- mutex, assuring that two mutexed actions must not be activated at the same time
- precondition of each action to ensure that it will be true if the action is activated
- fact, because it has to be supported by some active action to be true
- fact of initial state, to set it to true
- fact of goal state to require it to be true in a solution

Operations on $\mathcal{D}$ are then defined as follows:

$\mathcal{D} \oplus \langle a, l \rangle$ – a variable representing action a at layer l is required to be *true* (new formula added to the SAT representation)

$\mathcal{D} \ominus \langle a, l \rangle$ – a variable representing action a at layer l is required to be *false* (new formula added to the SAT representation)

`SinglePlan`$(\Pi, \mathcal{D})$ – a SAT solver is used to find a solution to the problem Π

Moreover, it is necessary to define how participant agent should handle a query from the initiator `AskAgent`(α, π). As described in caption of Algorithm 3, this query asks participant to assess the category the plan π – whether it is a prefix of *internally*, or *publicly extensible* plan. The participant first tries whether the provided plan is *internally extensible*, while it knows that the $\pi[1...(|\pi| - 1)]$ is *internally extensible*. Participant also stores information about internal actions that had to be inserted into this plan in order to mark it as *internally extensible*. It first match this subplan together with this information to its own planning graph $\mathcal{G}^{\alpha}$. Then, in first iteration, it tries whether the action $\pi[|\pi|]$ can directly follow previous actions by forcing this action to be used at appropriate layer. If it is not possible it tries this action at another layer allowing only internal action to be inserted in created gap. This continues until some limit ($l_{\max}$). If this process did not succeed then we know that this plan is not *internally extensible*. Similar procedure is used to detect *publicly extensible* plans.

Improvements

The described algorithm and its implementation can be improved in several ways. We use planning graph to create initial set of plans $\mathcal{D}^{l_{\max}}$. It can contain several actions at one layer, that are independent and can be executed in parallel. The initiator can query these actions in parallel. If all agents reply `CASE-I` then the initiator can continue with next action. If some agent replies `CASE-III`, the whole layer of action is forbidden in domain $\mathcal{D}$ and new plan has to be generated. If some agents reply `CASE-II`, their required action can be added into the plan in any order. It is also possible to only add actions required by one agent and continue with algorithm, because once it reaches the queried action again other agents will probably reply `CASE-III` again and then other required actions will be injected into the plan.

A participant replying with `CASE-III` can also take the initiative and continue the negotiation instead of the original initiator. This can be easily implemented in the case when only one agent is queried at a time. In the case of parallel queries, it is necessary to handle commitments of agents to different initiators to not promise excluding actions.

Experiments

For our experiments, we have used the *Tool Problem* (Tožička et al. 2014) that allows us to smoothly change an amount of public and internal actions between two extreme cases: (i) there is no internal action and (ii) there are as many internal actions as possible (implied by the equal sign in the definition of P^{pub}). Case (i) allows the initiator to construct a correct plan immediately without any communication, while case (ii) might require some negotiation. An advantage of case (ii) is, however, that initiator's plan is not so complex and contains less actions.

We have focused on the following three criteria: (1) number of SAT solver invocations by an initiator and participants, (2) the complexity of communication between the initiator and the participants (number of positive and negative responses), and (3) time complexity. We have measured these criteria as a function of the *publicness* of a problem, where the publicness is a number between 0 and 100 defined as $\frac{\text{\# public actions}}{\text{\# all actions}} * 100$. All our experiments have been carried out on CPU Intel Core-Duo 1.4GHz.

Tool Problem

In the *Tool Problem*, the goal requires that each of N agents performs its `doGoal` action. Nevertheless, in order to perform this action, agent need to `useTool` first. Then, there is an agent that provides the tools (`handTool`). This agent will be in the role of initiator. Initial state is that none of the agents has its tool and initiator has all of them. Goal state is that all tools have been used by actions `doGoal`.

In this problem, minimal publicness is 66.6% – all participants' actions `doGoal` have to be public because facts contained in goal are public and these actions have them as their effects; Initiator's actions `handTool` also have to be public because some of their effects are required by some other agents' actions; and actions `useTool` are internal. The maximal possible publicness is 100% when all actions

are public. In our experiments we continuously change the visibility of `useTool` actions and thus the publicness of the whole problem.

Results

Let us present results for the Tool Problem where $N = 5$. Each publicness settings was run 20 times. Another experiments for $N = 3$ and 4 yield similar results. All results are presented as a function of publicness of the problem (X-axis).

SAT Solver Invocations Figure 3 shows that with the increasing publicness the number of initiator's and participants' SAT Solver invocations decrease. The reason for it is that the more actions are available to the initiator the easier it is to find a local plan that all participants mark as *internally extensible*. In an extreme case, when all actions are public, the initiator performs only one SAT solver invocations and finds an *internally extensible* plan immediately.

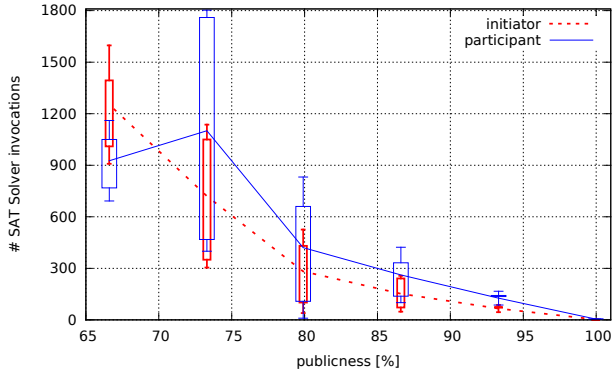


Figure 3: Average initiator's and participants' SAT Solver invocations depending on the problem publicness.

Number of `CASE-I` and `CASE-III` Replies The number of participants' `CASE-I` and `CASE-III` replies depending on the problem publicness is shown in Figure 4. It can be seen that in case of the minimal publicness (66%), the initiator creates many plans that participants reply with `CASE-III`. Note that the number of `CASE-I` replies first increases and then decreases with growing publicness. That is caused by the increase of relative number of `CASE-I` replies because a plan for one agent (whose actions are public) is correct, while the whole number of generated plans does not decrease significantly.

Time Complexity An average time of finding the solution depending on the amount of publicness is shown in Figure 5. Note that in the case of publicness of 66%, less time is required than in the case of publicness of 73%, although there are more SAT solver invocations in total. That is because the SAT problem is more constrained and it often does not have any solution, which can be often proved very easily by the SAT solver.

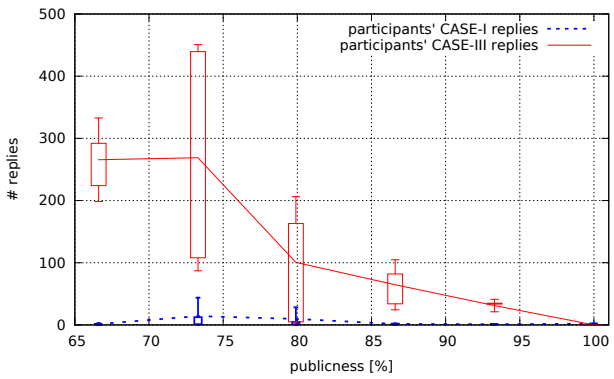


Figure 4: Average `CASE-I` and `CASE-III` replies depending on the problem publicness.

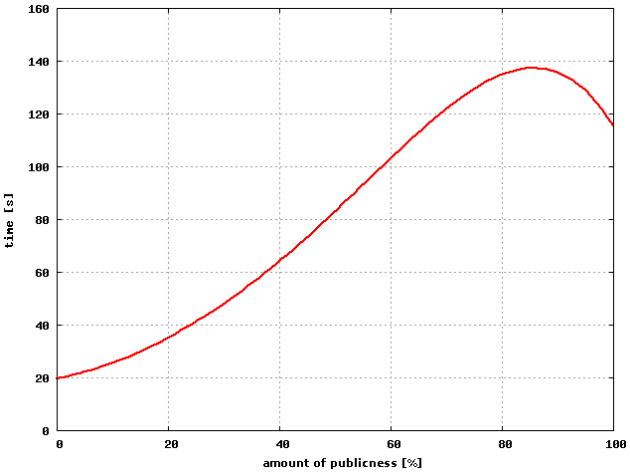


Figure 5: Amount of time required to solve the problem depending on the problem publicness.

Final remarks

The planning technique we proposed is carried out by a group of cooperative planning agents. Initially, each agent awaits a relevant part of a deterministic planning domain and a problem. After receiving the inputs, the agents firstly prepare their own local planning graphs based on the parts of the problem they have. Secondly, the agents use their individual SAT solvers to extract a local solution from their planning graphs. And finally, they negotiate actions from other agents to help them and not to interfere with the other agents by additional constraining of the SAT solving processes, forming a negotiation loop. If all the agents find a local plan and the plans provide all the requested goals without any conflicts, the planning process ends. Otherwise, the negotiation process continues until a solution is found.

Acknowledgements

This research was supported by the Czech Science Foundation (grant no. 13-22125S) and by a Technion fellowship.

References

Brafman, R., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS'08*, volume 8, 28–35.

Decker, K., and Lesser, V. 1992. Generalizing the Partial Global Planning Algorithm. *International Journal on Intelligent Cooperative Information Systems* 1(2):319–346.

Doherty, P., and Kvarnström, J. 2001. Talplanner: A temporal logic-based planner. *AI Magazine* 22(3):95–102.

Durfee, E. H. 1999. Distributed problem solving and planning. In Weiß, G., ed., *A Modern Approach to Distributed Artificial Intelligence*. San Francisco, CA: The MIT Press. chapter 3.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, 608–620.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Huang, R.; Chen, Y.; and Zhang, W. 2012. Sas+ planning as satisfiability. *J. Artif. Intell. Res. (JAIR)* 43:293–328.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of AAMAS'12*, 1265–1266.

Pellier, D. 2010. Distributed planning through graph merging. In Filipe, J.; Fred, A. L. N.; and Sharp, B., eds., *ICAART (2)*, 128–134. INSTICC Press.

Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *ECAI*, 762–767.

Tožička, J.; Jakubův, J.; Durkota, K.; Komenda, A.; and Pěchouček, M. 2014. Multiagent planning supported by plan diversity metrics and landmark actions. In *International Conference on Agents and Artificial Intelligence (ICAART)*.

Zhang, J. F.; Nguyen, X. T.; and Kowalczyk, R. 2007. Graph-based multiagent replanning algorithm. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, 122:1–122:8. New York, NY, USA: ACM.

Temporal Multiagent Planning with Concurrent Action Constraints

Matthew Crosby and **Ronald P. A. Petrick**
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
m.crosby@ed.ac.uk, rpetrick@inf.ed.ac.uk

Abstract

This paper investigates how centralised, cooperative, multi-agent planning problems with concurrent action constraints and heterogeneous agents can be encoded with some minor additions to PDDL, and how such encoded domains can be solved via a translation to temporal planning. Concurrency constraints are encoded on affordances (object-action tuples) and determine the conditions under which a particular object can (or must) be utilised concurrently. The effectiveness of the approach is evaluated on the Vehicles testing domain and on a new Warehouse domain, which is inspired by a real-world warehouse problem in which a centralised mission planner must find a concurrent plan for a fleet of robots in a manufacturing plant. The approach is shown to be promising, with the potential to support future work in the area.

Introduction

Planning in multiagent domains with concurrent actions—especially with complex concurrent action constraints—is not an easy task. With any meaningful number of agents and actions, the number of possible joint actions makes any direct planning approach infeasible. Furthermore, an efficient method for representing concurrency constraints is required as they are defined over the whole joint action space.

In previous approaches, concurrent action constraints are often defined on actions or unground action schema (Boutilier and Brafman 2001). While this technique provides a natural encoding method that specifies which *actions* can be performed simultaneously, it also leads to difficulties when dealing with problems that contain many objects that, while similar, have different concurrency properties. (For example, vehicles might carry different numbers of passengers, and doors might permit different numbers of agents to simultaneously pass through.) In domains such as these, it is useful to define concurrency constraints on the *objects* of the domain instead, leaving the features of PDDL that efficiently encode unground action schema untouched.

Given an encoding of concurrency constraints, the next task is to find plans that satisfy them. In this work, we consider an object-action representation of concurrency constraints and show that our encoding gives rise to an efficient translation into a temporal planning problem for which a solution is guaranteed to respect the concurrency constraints. The translation adds numeric fluents to the domain that keep track of which objects are being used, and also ensures that agents only perform one action at a time. We also show that, perhaps surprisingly, temporal planners can then be used to solve such problems in a reasonable time, even though there is no *a priori* reason to believe that they will be effective on the type of domains created by the translation process.

The motivating application for this work is a mission planning scenario for a fleet of heterogeneous robots in an assembly factory.[1] In this domain, the fleet of robots must navigate a factory floor and complete various picking and depalletising tasks in order to obtain a selection of parts as required for a kit. While one robot may be able to pick small, delicate objects, another may only be able to handle large, heavy objects. A centralised mission planner is in charge of finding and distributing plans for the robot fleet. As such, we treat this problem as an instance of centralised, completely cooperative, concurrent multiagent planning.

The capabilities of the robots are encoded in terms of *robot skills* (Bøgh et al. 2012), which are modelled at a level of abstraction above the robot control layer. Skill composition is traditionally done by hand, to create skill sequences that allow the robot to perform composite tasks. While this approach is satisfactory for controlled environments, by automating this process the robot will be better equipped to operate in continually-changing environments where the exact goals to be achieved are not necessarily known beforehand.

We model this domain as a classical planning problem, with additional concurrency constraints and agent action sets. The process of action execution is discretised into time steps with each action given an integer expected execution time. Currently, we only try to find a plan that assumes that the execution times are correct; we do not focus on methods for agent communication or the partial-order planning that would be required to deal with unknown execution times (Brenner 2003). In this work, plan synthesis and coordination are performed simultaneously so that a plan is generated both to achieve the goals and obey any concurrency constraints defined on the problem. Uncertainty about the envi-

[1] This work forms part of the EU STAMINA project. See http://stamina-robot.eu/ for more information.

ronment, and also about action execution success and duration, which are important features of the real-world problem, are left to future work and are not addressed in this paper.

The rest of this paper is structured as follows. We begin by discussing related work, broken down into four key topics: multiagent planning, temporal planning, robot skills, and affordances. The Warehouse domain is then introduced in order to motivate the approach taken in this paper. The details of the encoding are then presented, followed by the translation to temporal planning. Finally, we present the results of running a temporal planner on the translated domains, and conclude with a discussion of future work.

Related Work

We begin by discussing four strands of related work that are relevant to this paper: multiagent planning with concurrent actions, temporal planning, robot skills, and affordances.

Multiagent Planning with Concurrent Actions

As mentioned above, previous work on concurrent action constraints used the STRIPS (Fikes and Nilsson 1971) representation of actions, modified to include a concurrent action list describing the restrictions on the actions that could (or could not) be executed concurrently (Boutilier and Brafman 2001). This approach improved on previous work involving action interleaving which could not deal with truly concurrent interference effects. The work also showed how partial-order planning techniques could be used to solve such problems without the need for the introduction of an explicit notion of time. The proposed algorithm was not empirically evaluated but the work suggested that an interesting extension would be to utilise newer planning algorithms and techniques in solving such problems. Our work begins to do this, using a different encoding of constraints.

Previous work has also suggested encoding constraints on resources to specify which objects in the domain prohibit concurrent access (Knoblock 1994). We extend this idea to include resources which require concurrent access, and to include the possibility of resources that prohibit or require simultaneous access. While prior work has been interested in using the resource definitions to schedule concurrent actions to reduce overall plan execution time, we are more interested in encoding domains with inherent joint-action restrictions and planning over these restrictions.

Other relevant work considers joint actions in *strategic* multiagent planning (Jonsson and Rovatsos 2011). This work assumes the existence of an admissibility function that indicates which joint actions are possible, but does not discuss the details of such an encoding, instead simply assuming that an efficient encoding can be found depending on the features of a domain. A best-response planning (BRP) approach is introduced, that can find stable solutions to a certain class of planning problems called congestion games. Best response planning is well suited to plan refinement once a suitable starting plan has been found, but in our domains it is hard to find a suitable starting plan as this requires concurrent coordination from all the agents.

Temporal Planning

Temporal planning capabilities were introduced with PDDL 2.1 (Fox and Long 2003) and allow for the modelling of durative actions and the formulation of concurrent plans. In temporal planning problems, time is modelled explicitly, making temporal planning a natural fit for dealing with multiagent planning problems with concurrent actions. However, since (Brenner 2003) the approach has not been very prevalent in the recent multiagent planning literature.

In this paper we make use of POPF2 (Coles et al. 2010), a forward chaining partial-order planner which can find concurrent plans with low makespans. POPF2 was chosen because of its good performance—it was the runner-up in the temporal track of the 2011 International Planning Competition (Coles et al. 2012)—and because it has the capability to cope with all the constructs used in our translation.

Robot Skills

The encoding developed in this paper is motivated by the notion of *robot skills* (Bøgh et al. 2012), which have recently been proposed as an effective abstraction of the complex tasks that a robot can perform, and a tool for bridging the gap between low-level robot control and high-level planning.

In the taxonomy of robot skills, robot capabilities are separated into a three-level hierarchy consisting of motion primitives, skills and tasks. Motion primitives are the basic motion commands of the robot, at the lowest level of the hierarchy. Above this are robot skills, which represent the higher-level capabilities of the robot, which may require the use of many motion primitives. At the highest level are tasks, which can be scheduled without the need for specific robot control knowledge. It is at this level that planning occurs.

While the long-term goal of this work is to use automated methods for encoding planning actions from a description of robot skills, for this paper we hand-code each robot skill as a planning action, and then add separate skill distributions and concurrency constraints to the domain.

Affordances

Finally, we briefly consider the use of *affordances* in the related literature. The idea of an affordance can be traced back to Gibson (1977) and is a well-known concept in the robotics community and associated fields (see, e.g., (Duchon, Warren, and Kaelbling 1998; Lewis and Simó 2001; Steedman 2002; Sahin et al. 2007; Krüger et al. 2011), among others). For the purpose of this work, an affordance can be thought of as *the capacity of an object to be utilised in a certain manner*. We believe that affordances are an important component for encoding more complex concurrency constraints, and is a useful representational tool for building multiagent planning encodings. Affordances are discussed in more detail below in the context of encoding concurrency constraints.

The Warehouse Domain

We now describe the Warehouse domain, which models a problem in which a centralised mission planner is tasked with finding a concurrent plan for a heterogeneous robot fleet, working towards the shared goal of collecting sets of

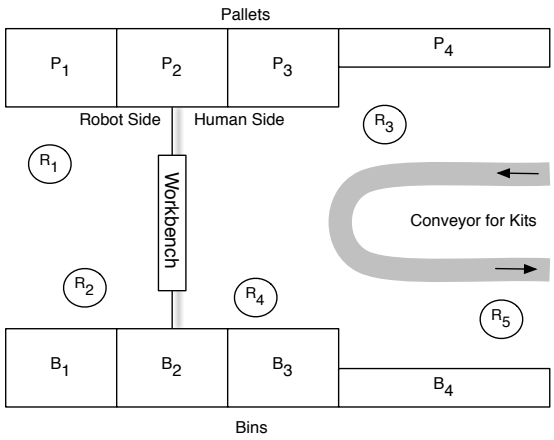


Figure 1: A depiction of the Warehouse domain. Some of the robots are only allowed on the robot side of the factory while others can navigate safely amongst humans. The aim is to pick given objects and place them in the correct kits which are located on the workbench and accessible to all robots. Once completed, the kits are carried to the conveyor and sent off for use in the manufacturing process.

parts (called kits) for use in a manufacturing plant. As mentioned above, the robots have a built-in representation of 'skills' which abstracts away from their low-level sensorimotor commands, and which denotes the capabilities that can be used to derive actions for planning purposes.

There are many different levels of abstraction at which the Warehouse domain can be modelled. For this initial investigation, we chose to focus on the features of concurrency constraints and heterogeneous agents, to test the feasibility of our encoding and of using temporal planning. This means that there are many factors that we do not cover that will nevertheless be important in a completely accurate representation of the real-world problem, including incomplete domain knowledge, action failure, and dynamic goals.

An example Warehouse problem is shown in Figure 1, which depicts the robots' operating environment. The goal of the domain is to assemble certain kits: collections of objects required for later use in the manufacturing process. The output should be a joint plan in which kits are put together efficiently and for which all the concurrency constraints are met so that, for example, no two agents attempt to access the same bin, pallet, or kit simultaneously.

The warehouse has been artificially split into two sections, one in which robots must navigate alongside humans, and the other in which only robots are allowed. Depending on the robot type, a robot may either be confined to the robot side or also allowed to work on the human side. A list of the skills that the each robot possesses, along with the time taken to perform each skill, is shown in Table 1. We assume that time can be discretised and that the actions always take the modelled time. In future extensions of this work, we will likely need to find partial-order plans that are robust to actions failing or taking longer than expected.

The pallets, found at the top of Figure 1, contain items that a robot can obtain if it has the depalletise skill. The bins at the bottom contain items that a robot can obtain if it can perform the requisite picking skill, either `pick_delicate` or `pick_heavy`, depending on the item in the bin. The complexities of robot navigation, and the picking, depalletising, and kitting processes are completely abstracted away by the robot skills formulation of our model, and assumed to be dealt with by low-level robot control processes.

Skill	Time	Robots with Skill
navigate-robot-side	2	R_1, R_2
navigate-human-side	3	R_3, R_4, R_5
pick-heavy	1	R_1, R_2, R_3
pick-delicate	1	R_1, R_3, R_4, R_5
depalletise	1	R_1, R_3, R_4
add-to-kit	1	R_1, R_2, R_3, R_4, R_5
deliver-kit	1	R_3, R_4, R_5

Table 1: Table showing the skills in the domain, their estimated duration, and the robots that can perform each skill.

While under completion, kits are placed on a workbench, accessible to all robots, but only one at a time. A finished kit must be carried to the conveyor, which requires two robots to carry the box simultaneously. While this last constraint is not part of the real-world problem, it is used to explore the ability of our approach to deal with concurrent coordination. The Vehicles domain presented later includes actions that require up to ten agents to coordinate concurrently.

Modelling and Encoding

This section discusses our method for encoding domains with properties such as those exhibited by the Warehouse domain. Our encoding is based on adding concurrency constraints to domains written in ordinary (non-temporal) PDDL (McDermott 2000). The features of the domains we would like to model include:

Multiagent Concurrent Actions: The domain contains multiple agents that execute their plans concurrently.

Cooperative Centralised Planning: The agents share the goal and a centralised mission planner plans for all agents.

Classical Planning Assumptions: All actions are deterministic, and there is complete domain knowledge with a fixed initial state and a known goal state (or set of states).

Agent Action Sets: Agents have different capabilities, the actions in the domain are split amongst the agents.

Concurrent Action Constraints: There are constraints over which actions can (or must) be performed simultaneously.

The problem that we are modelling is MA-STRIPS (Brafman and Domshlak 2008) (a multiagent extension of STRIPS) with the addition of concurrency constraints. More formally, we define a multiagent planning problem as a tuple $\Pi = \langle N, P, \{A_i\}_{i=1}^{n}, I, G, c\rangle$, where:

- $N = \{1, \ldots, n\}$ is the set of agents,
- P is the finite set of propositions of the domain,

- $I \subseteq P$ encodes the initial state,
- $G \subseteq P$ encodes the goal conditions,
- each A_i is agent i's action set, and
- $c : A_1 \times \ldots \times A_n \rightarrow \{0, 1\}$ specifies whether a particular joint action is valid under the concurrency constraints.

An action $a_i \in A_i$ has standard STRIPS syntax and semantics with $a_i = \langle pre(a), add(a), del(a) \rangle$ containing preconditions, add effects, and delete effects. A joint action $\bar{a} = (a_1, \ldots, a_n)$ is a member of the set $A = A_1 \times \ldots \times A_n$. For joint action $\bar{a}$, $pre(\bar{a}) = \bigcup_i pre(a_i), del(\bar{a}) = \bigcup_i del(a_i)$, and $add(\bar{a}) = \bigcup_i add(a_i)$. Joint action $\bar{a}$ is applicable in state s if and only if $del(\bar{a}) \cap add(\bar{a}) = \emptyset$, $c(\bar{a}) = 1$ (i.e., $\bar{a}$ satisfies the concurrency constraints), and $pre(\bar{a})$ holds in s. The application of a joint action updates the combined add and delete effects as in standard STRIPS.

A *plan* $\pi = [\bar{a}^1, \ldots, \bar{a}^k]$ is a sequence of joint actions such that $\bar{a}^1$ is applicable in the initial state I, and each subsequent joint action is applicable in the state resulting from the application of the previous action. It is assumed that each action set contains a no-op action with empty precondition and effects that can be used to align agents' plans.

As the domain of c is exponential in the number of agents, explicitly defining it (or even the set of joint actions) is not a practical approach. We therefore introduce an efficient method for encoding concurrency constraints by adding a construct to the PDDL problem file. A similar method is used to encode agent action sets for heterogeneous agents.

Encoding Agent Action Sets

We assume that the domain file contains a type `agent` which contains every member of N and no other objects. We also assume that each action has at least one parameter of type `agent`. A new construct, `capabilities`, is defined in the PDDL problem file which has the following (EBNF) syntax:

```
(:capabilities <cap-def>+)
<cap-def> ::= (<agent-name> <action-name>+)
```

where `agent-name` is a name of one of the agents, and `action-name` is a name of an action defined in the domain file. The intended interpretation is that the agent denoted by `agent-name` is capable of performing only the actions that appear in the list. In other words, the capabilities definition contains an agent and a list of action names that the agent can perform.

For example, for the Warehouse domain we would have:

```
(:capabilities
  (R1 navigate-robot-side pick-heavy
      pick-delicate depalletise add-to-kit)
  (R2 ...)
  ...
)
```

Using this method, it is possible to encode actions at the domain level without having to worry about which agents can perform each action in a particular problem instance.[2]

[2] As there are many problems for which some (or all) agents can perform all actions we assume that a missing capabilities definition means that all agents are capable of performing every action in the domain. We also assume that any agents not included in the capabilities definition have the ability to perform all actions.

Encoding Concurrency Constraints

At first glance, it may seem that concurrency constraints should be defined along with action specifications, since they constrain which actions can be performed concurrently. However, considering the type of domains we would like to encode, it makes more sense to associate constraints with objects. In particular, a standard action schema may have different concurrency constraints depending on the size, shape, or capacity of the object that it is ground to. (For example, consider a domain with multiple types of vehicles, or network connections of varying bandwidth.)

While the next obvious step might be to define concurrency constraints directly on the objects themselves, in the general case, this is not expressive enough. For instance, consider the case of an object that can be used in multiple different ways: a door might only be *passed through* by a single agent at a time, yet can be simultaneously *painted* by multiple agents. There are countless possible examples of *different ways* an object can be used, all of which potentially affect concurrency constraints. We call these different ways an object can be used its *affordances*.

Affordances are clearly related to combinations of objects and actions, but the exact details of this relationship are not obvious. A further complication comes from the fact that an affordance of an object may be associated with multiple actions. For example, a claw hammer may have the associated actions of `bash_nail` and `extract_nail`. However, in terms of a planning problem we may only want to constrain the higher level affordance of the hammer to be *used as tool*.

In order to deal with the preceding cases we define constraints over object-action tuples. Each tuple consists of an object and a list of actions that can be thought of as utilising the object for a particular affordance. This way an object can have multiple affordances (when it appears in different constraints with different lists of actions) and an affordance can be utilised by multiple actions (an action list is used).

We encode concurrency constraints as follows:

```
(:concurrencies <conc>+)
<conc> ::=
    (<obj-name> <act-name>+ <min> <max>)
```

where `obj-name` is an object from the problem definition, `act-name` is an action name from the domain file in which `obj-name` appears in a possible grounding, and `min` and `max` are positive integers with `max` $\geq$ `min`. The intended interpretation of the concurrency constraints is that 'not more than `max` and at least `min` agents can simultaneously utilise object o via the actions in the action list'. A concurrency constraint of `(o act 1 n)` therefore effectively provides no constraint on the object o.

We expect that all objects of a particular type share in their relevant affordances stipulating that if a concurrency constraint exists for object o of type t with action list $\bar{a}$, then a concurrency constraint exists with action list $\bar{a}$ for each object of type t. This can be easily achieved by adding 'dummy' affordances with $min = 1$ and $max = n$. We then

define the affordances of the domain as each pair (type, action list) for which a ground object of that type appears with that action list in the concurrencies specification.

The following example shows a selection of concurrency constraints for the Warehouse domain:

```
(:concurrencies
  (bin1 pick-delicate pick-heavy 1 1)
  (pallet1 depalletise 1 1)
  (kit1 deliver-kit 2 2)
  ...
)
```

In particular, it specifies that if a `pick-delicate` or `pick-heavy` action is applied to `bin1` then no other agent can concurrently perform a `pick` action on `bin1`. It also says that the action `deliver-kit`, when ground to `kit1`, must be performed by exactly two agents concurrently. More complicated concurrency constraints will be discussed in the evaluation section when we look at the Vehicles domain.

It should be noted that there are certain nuances to defining concurrency constraints in this way. Concurrency constraints specify constraints on the simultaneous execution of actions, not facts about what can be true over any state in the world. So, for example, modelling that two robots cannot be in the same location should be part of the domain definition as it is a fact about which possible states are allowed.

Translation to Temporal Planning

At this point, we have a domain written in classical single-agent PDDL, with concurrency constraints and capability definitions added to the problem file. The next step is to try and find a plan that obeys the intended semantics of the capabilities and concurrency constraints. To do this, we translate our encoded domains into PDDL 2.1 with durative actions and numeric fluents, for use with a temporal planner.

Pseudocode for the translation is shown in Algorithm 1, which is split into three steps: adding capabilities, translating to durative actions and adding concurrency constraints. Along with the introduced encoding, the translation assumes that the domain definition includes an *agent* type and there are at least two objects of this type, each action has an associated duration and all objects in the domain are typed.

Adding Capabilities

Adding capabilities is straightforward and does not require any temporal constructs and as such is performed before the translation to durative actions. For each action `act` that appears in the capabilities definition, a new `(can-act ?a - agent)` predicate is added to the list of predicates. Then, each action that appears gains an additional precondition `(can-act ?x)` for each parameter `?x` of type agent. This precondition ensures that any agent the action is ground to has capability of performing the action. Finally, for each agent `a` defined as capable of performing `act`, the static proposition `(can-act a)` is added to the initial state.

If an action does not appear in the capabilities, then it does not get modified, which means that it can be ground to any agent and therefore any agent may perform the action. If no capabilities are defined, then nothing is changed in this step and it is possible for all agents to perform each action.

Algorithm 1: Translate to temporal planning problem.

Input : PDDL domain and problem files (dfile, pfile)
Output: PDDL temporal problem and domain files

```
   // Add Capabilities
1  foreach act in :capabilities do
2      dfile.predicates.add(can-act ?a - agent)
3      foreach predicate ?a of act of type agent do
4          dfile.act.addprecondition(can-act ?a)
5      foreach agent in :capabilities containing act do
6          pfile.init.add(can-act agent)
   // Translate to durative actions
7  dfile.requirements.add(:fluents)
8  dfile.predicates.add(free ?a - agent)
9  foreach act in dfile with duration d do
10     action ← durative-action
11     act.add(= ?duration d)
12     precondition ← condition
13     foreach condition in act do
14         condition ← (at start (condition))
15     foreach effect in act do
16         if effect is positive then
17             effect ← (at end (effect))
18         else
19             effect ← (at start (effect))
20     foreach agent parameter ?a in act do
21         act.conditions.add(at start(free ?a))
22         act.effects.add(at start(not (free?a)))
23         act.effects.add(at end(free ?a))
   // Add concurrency constraints
24 foreach affordance with type t and action list ā do
25     dfile.functions.add(using-t-ā ?o - t)
26     dfile.funcitons.add(min-t-ā ?o - t)
27     dfile.functions.add(max-t-ā ?o - t)
28     foreach act in ā do
29         if max > 1 then
30             copy act to new act-join
31             act-join.con.add(at start (> (using-t-ā ?o) 0))
32             act-join.eff.add(at start (increase (using-t–ā ?o)
               1))
33         act.con.add(at start (= (using-t–ā ?o) 0))
34         act.con.add(at end (>= (using-t-ā ?o) (min-t-ā ?o)))
35         act.con.add(at end (<= (using-t-ā ?o) (min-t-ā ?o)))
36         act.eff.add(at start(increase (using-t-ā ?o) 1))
37         act.eff.add(at end(assign (using-t-ā ?o) 0))
38         act ← act-start
39 foreach concurrency constraint (o, ā, min, max) do
40     pfile.init.add(= (using-t-ā o) 0)
41     pfile.init.add(= (min-t-ā o) min)
42     pfile.init.add(= (max-t-ā o) max)
```

Translating to Durative Actions

The next step of the algorithm is to translate all actions into durative actions for use in temporal planning. This step also adds a predicate `(free ?a)` that is used to make sure that each agent only ever performs a single action at a time. This predicate is removed whenever an agent starts an action and is only replaced on completion of the action (lines 20–23 in Algorithm 1). Action durations equal to those specified for each action are also added at this stage.

By convention, we assume that all preconditions are

required to be met at the beginning of the durative action, and they are therefore all changed to `(at start (condition))`. We set negative effects to be updated at the beginning of an action while positive effects occur on completion of the action. While it is certainly possible to design a domain for which this is a non-natural interpretation, it works well for the domains we have used and we assume that the domain creator is aware of this fact.

Adding Concurrency Constraints

The final part of the algorithm (lines 24–42) is the most involved and requires the addition of new actions and fluents. For each action, the translation creates at most two new temporal actions, 'action–start' and 'action–join'. For each different affordance, three new functions must be added to the domain: `using`, `min`, and `max`. The first of these is updated by the corresponding actions to show how many agents are currently utilising a constrained affordance. The latter two are used to ensure that the number of agents that simultaneously use a constrained affordance is between the minimum and maximum specified for that resource.

The first part of the translation shown in Algorithm 1 (lines 30–32) involves the creation of the new 'join' action for each action that appears in a concurrency constraint with $\texttt{max} > 1$. The additional elements for the join action will be explained after we have discussed the 'start' action. It appears first in the algorithm only because it is built from a copy of the unmodified start action.

To each start action, the following is added to the condition for each constrained parameter `?p`:

```
(at start (=  (using-p-ā ?p)  0))
(at end   (>= (using-p-ā ?p) (min-p-ā ?p)))
(at end   (<= (using-p-ā ?p) (max-p-ā ?p)))
```

This means a joint action can only be started if there are currently no agents already engaged in it, and by the end of the action the number of agents engaged with the related object is between the maximum and minimum values. The following is added to the effects for each constrained parameter:

```
(at start (increase (using-p-ā ?p) 1))
(at end   (assign   (using-p-ā ?p) 0))
```

This updates the number of agents utilising the relevant affordance of `?p` and then resets it once the action is complete.

The join actions are simpler. To each join action the following is added to the condition:

```
(at start (> (using-p-ā ?p) 0))
```

This ensures that a join action can only be performed if the start action is currently initiated. The following is added to the actions effects:

```
(at start (increase (using-p-ā ?p) 1))
```

which is simply the same counter used in the start action.

Putting this all together, we end up with a start action for each constrained action and a join action only for actions that have concurrency constraints that allow multiple agents (i.e., $\texttt{max} \geq 2$). There is no 'end' action needed as the conditions of the start action ensure that the correct number of agents are performing the action by the time it has finished. When run with a temporal planner, a joint action is formed from all actions scheduled at a particular time step. It is possible, for actions with long durations, that the join action does not appear in the same timestep as the start action (which violates the intended interpretation). However, the planner used in this work, POPF2, always schedules join actions immediately after start actions as it attempts to minimise makespan, making the above translation sufficient.

Optimisations

While the previous algorithm creates a functionally correct domain, there are many optimisations that can be performed based on the type of concurrency constraints that exist in the problem. We introduce a few of them here and then examine their effects on planning in the evaluation section below.

Actions that require two agents to perform them simultaneously can be encoded directly with two agent parameters, and by including the condition `(not (= ?a1 ?a2))` to ensure that the parameters cannot be ground to the same agent. However, this is not good practice in the general case (where n agents are required to perform the action) as, firstly, it requires $(n(n+1))/2$ inequality clauses and, secondly, this method does not allow the grounding of actions to objects with different types of concurrency constraints. If a concurrency constraint of `(o a 2 2)` appears for all groundings of an affordance, instead of following the algorithm, we use the equality definition just mentioned.

It is also possible to optimise the translation for concurrency constraints of the form `(o a 1 1)`. These constraints are quite common, as there are many objects that can only be used by one agent at a time. Notice that the constraint `(o a 1 1)` is similar to the constraint that each agent may only perform one action at a time, which can be encoded using a `(free ?a)` predicate. The same method can be used for objects, with a `(free-o ?o)` predicate in place of the agent one. A final small optimisation can be applied when an action appears in each agent's capabilities list. In this case, the associated `(can-act ?a)` predicate does not need to be added to the domain (i.e., lines 2-5 in Algorithm 1 can be skipped).

Evaluation

This section presents the results of running a temporal planner on the translated domains. A python script was written to perform the translations, including the optimisations mentioned above.[3] The planner we chose for this work was POPF2 (Coles et al. 2010), since it has the capability to cope with all the constructs used in the translation, and also to produce plans that attempt to minimise makespan. POPF2 was also the best planner in IPC11 at dealing with temporal problems with inbuilt concurrencies. All experiments were run on the same machine with 48GB of memory and a 2.66GHz processor. Experiments were allowed to run until a plan was found or an out of memory error was reported.

[3] Available for download at `http://homepages.inf.ed.ac.uk/mcrosby1`.

	Small		Medium		Large	
Kits	time(s)	span	time(s)	span	time(s)	span
1	0.02	21	0.03	12	0.14	8
2	0.78	49	1.15	32	0.93	18
3	8.01	87	9.71	53	74.5	24
4	30.7	105	37.5	62	–	–
5	106	139	133	76	–	–

Table 2: Table showing the time in seconds and makespan of the first plan found by POPF2 for different domain sizes and number of kits/goals.

Warehouse

This section presents an evaluation of the Warehouse domain. The first aim was to ensure that the planner could indeed solve problems of the scale we are likely to deal with in real-world settings. Three problem instances were created—small, medium and large—with medium having the same setup as depicted in Figure 1 and the robot capabilities shown in Table 1. The small domain involved just the right-hand side of Figure 1, with three robots, two bins (one containing a delicate item and one containing a heavy item), and two pallets. The only robots left were R_3, R_4, and R_5, which had the same skills as in the previous domain. The large domain contains ten robots, eight pallets, and eight bins. The five new robots added were given the same skills as those for R_1 to R_5, respectively, and the new pallets and bins were spread evenly across the two sides.

Table 2 shows how the planner performed over the different sizes of domains as the number of kits that needed to be delivered was changed. The goals were to deliver from one to five kits, each containing five separate parts. The goals were kept the same across the problems so that they could be compared directly in terms of time and makespan.

In the medium and large problems, the extra robots mean that more actions can be scheduled simultaneously so that the makespan can potentially be lower to achieve the same goals. It was unknown as to whether the fact that the problem was easier (due to more available robots) or that the problem was harder (due to a massively increased state space) would dominate the results. While being almost twice the size, the medium domain was solved in times that were not much slower than the small domain, showing that, perhaps surprisingly, increasing the number number of agents capable of performing tasks can somewhat counteract the increase in domain size. As can be seen from the table, the number of kits had a large effect on the planning times while increasing the number of robots and bins had a lesser effect.

While the final two large problems were unsolvable, the planner performed much better than expected given the number of joint actions, and the way they were encoded (not designed for ease of planning). For example, in the initial state of the medium problem each robot can perform eight or nine valid actions, meaning that there are over forty-six thousand possible joint actions. Obviously, the temporal planner does not deal directly with the joint action space, which is why it is a feasible approach for solving these kind of problems in the first place.

	No-Deliver		Equal		Robot-side	
Ag's	time(s)	span	time(s)	span	time(s)	span
3	0.97	65	0.97	70	0.97	70
4	4.25	41	4.82	42	4.82	42
5	8.18	37	10.83	36	4.71	29
6	6.64	26	15.82	24	8.02	28
7	14.06	23	26.16	28	–	–
8	35.25	23	56.77	26	14.4	41
9	139	24	210	22	51	39
10	43.25	21	70	20	26	39

Table 3: Table showing the time in seconds and makespan of the first plan found by POPF2 for different domain sizes and number of agents.

Given the previous results, we wanted to test how adding robots (but not otherwise increasing the problem size) affected planning. We used the middle problem from the previous table (medium size, 3 kits) as our starting point and only varied the number of agents. All agents were given every capability, except that half were confined to the robot side and half to the human side. The smallest problem instance contained 3 robots with two on the human side to ensure that the problem has a solution (i.e., the `deliver-kit` action requires two agents acting simultaneously).

The results of these experiments are shown in Table 3. The middle column shows the set-up described above and contains an anomalous looking result for the 9 agent problem. As the goal does not change over these problems, the makespan becomes closer to optimal as more agents are added (as there are more 'free' robots to assign actions to at each point). The time taken for the 9 robot problem appears to break the trend of the slow increase in planning times as the number of agents is increased. We hypothesised that this could be due to the `deliver-kit` action having exponentially more possible groundings now that another robot has been added to the human side. We therefore ran experiments without the `deliver-kit` action (column 1) and experiments where robots were only added to the robot side (column 3). From the results we can see that the `deliver-kit` action, while problematic, is not the sole cause of the anomaly. We can only conclude that due to the very large state space, there is noise in the results depending on the path of the heuristic search. This can also be seen by the fact that no result was found for the seven agent problem in column 3 even though one clearly exists.

Finally, we wanted to test the effectiveness of our translation optimisations. Table 4 shows the effects that the optimisations have on both planning times and plan cost. The reported values are of the form unoptimised/optimised (from Table 2) so that a value less than one represents an improved time or makespan. The cells containing 'x' show where the optimised version found a solution where the unoptimised version could not. The values of the table that contain '–' are cases where the planner did not manage to find a solution on the unoptimised translation. We can see that the coverage for the unoptimised case is much worse than for the optimised case. This means that the optimisations are nec-

	Small		Medium		Large	
Kits	time	span	time	span	time	span
1	333	**0.86**	3.33	1.08	1.86	1.13
2	14.7	**0.94**	22.3	1.03	120	1.06
3	1.20	**0.89**	7.97	**0.85**	x	x
4	2.71	**0.91**	x	x	–	–
5	x	x	x	x	–	–

Table 4: A comparison of the unoptimised translation with the results for the optimised translation. Each table entry is calculated as the value of unoptimised/optimised, meaning that the results show how much slower/more costly the unoptimised version was. A value of x shows where the unoptimised version did not find a plan but the optimised version did, whereas '–' represents that both translations failed.

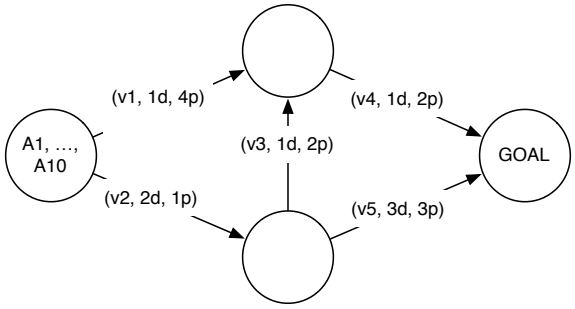


Figure 2: An example Vehicles problem. Ten agents (only six of which can drive), starting on the left, must travel to the rightmost location via a collection of vehicles requiring different numbers of drivers, and permitting different numbers of passengers. This problem is solved in 0.15s by POPF2.

essary, especially in the larger domains. It is also interesting to note that when the unoptimised domain returns a solution it is often of better quality than that of the unoptimised domain. However, coverage is a much more important factor as the planner can always be run for longer if a higher quality solution is required.

Vehicles

Finally, we test some further capabilities and properties of our approach on the Vehicles domain, first introduced in (Crosby 2014). In the domain, agents must use different vehicles to reach a particular goal location. The vehicles in the domain have associated concurrency constraints, such that one vehicle may require a driver and may hold up to four passengers, while another may require two drivers and not be able to hold any passengers. Some agents in the domain are designated as drivers and able to perform the `drive` action, while others are only able to perform the `passenger` action. An example problem is shown in Figure 2.

The Vehicles domain shows that it is possible to model conjunctive concurrency constraints with our approach, e.g.:

```
(and
  (v1 drive 1 1)
  (v1 passenger 0 4)
)
```

	Fig 2		(v, d1, p1)		(v, d5, p5)	
Drivers	time	span	time	span	time	span
5	–	–	–	–	0.03	2
6	0.15	3	–	–	0.15	2
7	–	–	0.64	5	2.42	2
8	0.32	4	1.1	5	–	–
9	–	–	2.2	6	–	–
10	3.13	5	1.99	6	–	–

Table 5: Results for the Vehicles domain as the number of drivers increases. The first column is for the problem shown in Figure 2 while the latter two are for the same problem with all vehicle constraints replaced with that shown.

The intended interpretation of this constraint is that the vehicle $v1$ can have at most one concurrent driver and simultaneously up to four passengers. This is translated to the temporal planning encoding by adding the condition:

```
(at start (> (using-drivable ?v ) 0))
```

to the passenger action. This method can be used for any conjunctive constraint by designating one element (with `min` > 0) as the initial action, and adding the relevant condition to all other actions. However, it is not yet known if this works for problems where the concurrency constraints overlap in terms of the actions they include.

The results of running our algorithm on the Vehicles domain are shown in Table 5. Interestingly, the results were very dependent on the number of driver agents in the domain, presumably because certain numbers of drivers lead to dead ends early in the problem. Overall, the coverage was not very impressive while the planning times when a solution was found were surprisingly fast. We take the optimistic view that the results are promising in that future work can build on the areas where temporal planners are effective for these type of problems.

Conclusion

This paper presented a method for encoding and solving planning problems with concurrent interacting actions and heterogeneous agents. For simple concurrency constraints, where the translation can be optimised, the approach was shown to be effective. However, more work is needed to plan with more complex concurrency constraints.

In the future, we plan to analyse further the conditions under which the temporal approach is effective, and use this as a starting point for creating a planning algorithm specifically designed for domains with concurrency constraints. We also intend to explore the notion of affordances for multiagent planning further and see how different representations can be utilised in the planning process. Finally, we intend to introduce some further complexities of the real-world Warehouse domain into our work.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant agreement no. 610917 (STAMINA).

References

Bøgh, S.; Nielsen, O. S.; Pedersen, M. R.; Krüger, V.; and Madsen, O. 2012. Does your Robot have Skills? In *Proceedings of the International Symposium on Robotics (ISR 2012)*.

Boutilier, C., and Brafman, R. 2001. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research* 14:105–136.

Brafman, R. I., and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 28–35.

Brenner, M. 2003. A Multiagent Planning Language. In *Proceedings of the Workshop on PDDL at the International Conference on Automated Planning and Scheduling (ICAPS 2003)*.

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 2010)*, 42–49.

Coles, A. J.; Coles, A.; Olaya, A. G.; Celorrio, S. J.; López, C. L.; Sanner, S.; and Yoon, S. 2012. A Survey of the Seventh International Planning Competition. *AI Magazine* 33(1):83–88.

Crosby, M. 2014. A Temporal Approach to Multiagent Planning with Concurrent Actions. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2013)*.

Duchon, A. P.; Warren, W. H.; and Kaelbling, L. P. 1998. Ecological robotics. *Adaptive Behavior, Special issue on biologically inspired models of navigation* 6(3-4):473–507.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20:61–124.

Gibson, J. 1977. The theory of affordances. In Shaw, R., and Bransford, J., eds., *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, 67–82.

Jonsson, A., and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 114–121.

Knoblock, C. A. 1994. Automatically Generating Abstractions for Planning. *Artificial Intelligence* 68:243–302.

Krüger, N.; Geib, C.; Piater, J.; Petrick, R.; Steedman, M.; Wörgötter, F.; Ude, A.; Asfour, T.; Kraft, D.; Omrčen, D.; Agostini, A.; and Dillmann, R. 2011. Object-Action Complexes: Grounded abstractions of sensory-motor processes. *Robotics and Autonomous Systems* 59(10):740–757. doi:10.1016/j.robot.2011.05.009.

Lewis, M. A., and Simó, L. S. 2001. Certain principles of biomorphic robots. *Autonomous Robots* 11(3):221–226.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* 21(2):35–55.

Sahin, E.; Çakmak, M.; Doğar, M. R.; Uğur, E.; and Ücoluk, G. 2007. To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior* 15(4):447–472.

Steedman, M. 2002. Plans, affordances, and combinatory grammar. *Linguistics and Philosophy* 25(5-6):723–753.

A Privacy-preserving Model for the Multi-agent Propositional Planning Problem

Andrea Bonisoli and **Alfonso E. Gerevini** and **Alessandro Saetti** and **Ivan Serina**
Università degli Studi di Brescia, email:{firstname.surname}@unibs.it

Abstract

Over the last years, the planning community has formalized several models and approaches to multi-agent (MA) planning. One of the main motivations in MA planning is that some or all agents have private knowledge that cannot be communicated to other agents during the planning process and the plan execution. In this paper, we propose a model that preserves the privacy of the involved agents, and discuss how the MA-A^* search algorithm can be adapted to implement our model.

Introduction

Over the last years, the planning community has formalized several models and approaches to multi-agent (MA) planning (e.g., (Brafman, & Carmel 2008; Nissim, & Brafman 2012; Torreño, Onaindia, & Sapena 2012)). One of the main motivations in MA planning is that some or all agents have private knowledge that cannot be communicated to other agents during the planning process and the plan execution.

The most known model for the MA planning task is MA-STRIPS (Brafman, & Carmel 2008). In MA-STRIPS, the set of the executable actions is partitioned into n sets $\{A_i\}_{i=1}^{n}$, such that A_i is the set of actions the i-th agent is capable of executing. A proposition is considered *private* if it is required and affected only by the actions of a single agent. All other propositions are considered *public*. An action is private if all its preconditions and effects are private; the action is considered public, otherwise. An efficient approach (Nissim, & Brafman 2012) using MA-STRIPS is the multi-agent (distributed) formulation of A^* (MA-A^*). In MA-A^*, no agent has complete knowledge of the search state, and hence during the A^* search each agent sends a message to the other agents including a representation of the state under expansion, where, for sake of the agents' privacy, private propositions are encrypted.

The approach described in (Nissim, & Brafman 2012), which uses the MA-STRIPS formulation of MA planning, does not fully guarantee the privacy of the involved agents when: at least one public proposition is confidential (i.e., it should be kept hidden from some agent), or the identity/existence of at least one agent is confidential, and hence only certain authorized agents

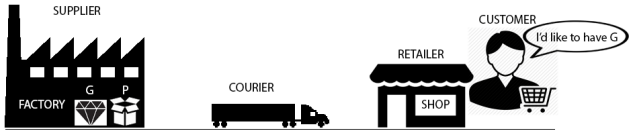


Figure 1: An instance of a multi-agent scenario.

can communicate with her. For instance, consider the scenario depicted in Figure 1. In this scenario, four agents act: the retailer (`Re`), the courier (`Co`), the retailer's supplier (`Su`), and the retailer's customer (`Cu`). Assume that customer `Cu` needs to have goods `G` that are not currently in the retailer's shop (`Sh`). Retailer `Re` sends a purchase order to its supplier `Su` for the shipment of a package `P` containing `G`. Express delivery courier `Co` moves package `P` from the supplier's factory (`F`) to the retailer's shop. Assume also that (`pack G P F`) is an action of agent `Su`; (`load T P F`) is an action of agent `Co`; (`unpack G P Sh`) is an action of agent `Cu`; (`in G P`) and (`loadable P`) are two (positive) effects of (`pack G P F`); (`loadable P`) is a precondition of (`load T P F`); and, finally, (`in G P`) and (`at P Sh`) are two preconditions of (`unpack G P Sh`). Essentially, actions (`pack G P F`), (`load T P F`), and (`unpack G P Sh`) represent, respectively, that at factory `F` supplier `Su` packs goods `G` in package `P`, making package `P` loadable, courier `Co` loads package `P` from `F` into truck `T`, and, at shop `Sh`, customer `Cu` unpacks goods `G` from package `P`.

According to the approach described in (Nissim, & Brafman 2012), for this scenario propositions (`in G P`) and (`loadable P`) are public; hence, when agent supplier `Su` communicates the state obtained by executing its action (`pack G P F`) to agent courier `Co`, proposition (`in G P`) is not encrypted. The negative effect of this information exchange is that the privacy of the customer is violated, as it makes the courier know the content of package `P`. In order to preserve the privacy of the involved agents, the supplier should not communicate (`in G P`) to the courier. Moreover, while proposition (`in G P`) needs to be somewhat communicated to the retailer's customer so that the customer will be able to unpack goods `G` from package `P`, there should be no

(direct) contact between the retailer's supplier and the retailer's customer.

In MA-STRIPS, the privacy of the involved agents may also be violated by the definitions of the initial state and the set of problem goals, because these definitions are shared among the involved agents. As for our scenario, using MA-STRIPS, every agent knows that initially goods `G` and package `P` are at the supplier's factory, and that customer needs to have goods `G`. This violates the privacy of the retailer's supplier and the retailer's customer.

In this paper, we propose a model that preserves the privacy of the involved agents, and, discuss how the MA-A^* search algorithm can be adapted to implement our model.

The model of MA planning that is most similar to the one we propose here is the model adopted by MAP-POP (Torreño, Onaindia, & Sapena 2012). MAP-POP is a multi-agent planning system searching the space of partial-order plans by an A^* POP algorithm. Each agent selects a (partial) plan π from an open list, chooses an open (sub)goal g from the selected plan, computes a set of plans refining π to achieve g, sends the refined plans to every other agent, and receives the plans refined by other agents. This exchange of (partial) plans can imply some violation of the agents' privacy. For the example of MA planning described above, according to the MAP-POP's approach, the customer agent computes the (partial) plan consisting of action (`unpack G P Sh`) in order to refine the initial empty plan for achieving the customer's goal, and sends the refined plan to the supplier agent (by occluding precondition (`at P Sh`)). However, as discussed before, for this scenario no contact between the retailer's supplier and the retailer's customer should be established. Our model of MA planning avoids the global broadcasting and, by restricting message passing to certain agents, guarantees that, even when the plan is executed, the identity/existence of certain agents remains confidential.

Privacy-preserving Multi-agent Planning

A *privacy-preserving multi-agent planning problem* for a set of agents $\Sigma = \{\alpha_i\}_{i=1}^n$ is a tuple $\langle \{A_i\}_{i=1}^n, \{F_i\}_{i=1}^n, \{I_i\}_{i=1}^n, \{G_i\}_{i=i}^n, \{M_i\}_{i=1}^n \rangle$ where:

- A_i is the set of actions agent α_i is capable of executing, and such that for every pair of agents α_i and α_j, $A_i \cap A_j = \emptyset$;
- F_i is the set of relevant facts for agent α_i;
- $I_i \subseteq F_i$ is the portion of the initial state relevant for α_i;
- $G_i \subseteq F_i$ is the set of goals for agent α_i;
- $M_i \subseteq F_i \times \Sigma$ is the set of messages agent α_i can send to the other agents.

Facts and actions are literals and pair $\langle Pre, Eff \rangle$, respectively, where *Pre* is a set of positive literals and *Eff* is a set of positive or negative literals. Let $X+/X-$ denote the positive/negative literals in set X, respectively. Let $\mathcal{G}$ be the graph induced by $\{M_i\}_{i=1}^n$, where nodes represent agents, and edges represent possible information exchanges between agents; i.e., an edge from node α_i to node α_j labelled p represents the agent α_i's capability of sending p to agent α_j. In order to have well-defined sets $\{M_i\}_{i=1}^n$, $\forall \alpha_i, \alpha_j \in \Sigma$, $\forall p$ s.t. $p \in F_i$ and $p \in F_j$, there should be a path in $\mathcal{G}$ from the node representing α_i to the node representing α_j formed by edges labelled p, if $p \in I_i$, or $\exists a \in A_i \cdot p \in Eff+(a)$, or $\exists a \in A_i \cdot p \in Eff–(a)$.

A plan for a multi-agent planning problem is a set $\{\pi_i\}_{i=1}^n$ of n single-agent plans. Each single agent plan is a sequence of happenings. Each happening of agent α_i consists of a (possibly empty) set of actions of α_i, and a (possibly empty) set of exogenous events. Exogenous events are facts that become true/false because of the execution of actions of other agents; in this sense, these events cannot be controlled by agent α_i. Formally, $\pi_i = \langle h_i^1, \ldots, h_i^l \rangle$, $h_i^j = \langle A_i^j, E_i^j \rangle$, $A_i^j \subseteq A_i$, $E_i^j \subseteq \bigcup_k F_k$, for $i = 1 \ldots n$, $j = 1 \ldots l$, $k \in \{1, \ldots, i-1, i+1 \ldots, n\}$.

The execution of plan π_i generates a state trajectory, $\langle s_i^0, s_i^1, \ldots, s_i^l \rangle$, where $s_i^0 = I_i$, and a sequence of messages, $\langle m_i^1, \ldots, m_i^l \rangle$, each of which is a set of literals. At planning step j agent α_i sends literal $p/\neg p$ if either α_i executes an action that makes p true/false or α_i receives the message that lets the agent know p becoming true/false. In this latter case, α_i forwards the received message $p/\neg p$ to the agents it is connected to. For every planning step, the forwarding is repeated $n-1$ times so that, if sets $\{M_i\}_{i=1}^n$ are well-defined, every agent α_k such that $p \in F_k$ is advised that p becomes true or false (the length of the shortest path between any pair of nodes in the graph induced by $\{M_i\}_{i=1}^n$ is at most $n-1$).

Formally, state s_i^j and message m_i^j are defined as follows, for $j = 1 \ldots l$ and $k = 1 \ldots i-1, i+1 \ldots n$.

$$s_i^j = s_i^{j-1} \cup \bigcup_{a \in A_i^j} Eff+(a) \cup E+_i^j \setminus \bigcup_{a \in A_i^j} Eff–(a) \setminus E-_i^j;$$

$$m_i^j = \bigcup_k sm+_{i \to k}^j (n-1) \cup \bigcup_k sm-_{i \to k}^j (n-1), \text{ with}$$

$$sm+_{i \to k}^j (t) = \Big\{ \langle p, \alpha_k \rangle \mid \langle p, \alpha_k \rangle \in M_i, \; p \in \bigcup_{a \in A_i^j} Eff+(a) \cup rm+_i^j (t-1) \Big\},$$

$$sm-_{i \to k}^j (t) = \Big\{ \langle \neg p, \alpha_k \rangle \mid \langle p, \alpha_k \rangle \in M_i, \; p \in \bigcup_{a \in A_i^j} Eff–(a) \cup rm-_i^j (t-1) \Big\},$$

$$rm+_i^j (t) = \Big\{ p \mid \langle p, \alpha_i \rangle \in \bigcup_k sm+_{k \to i}^j (t) \Big\},$$

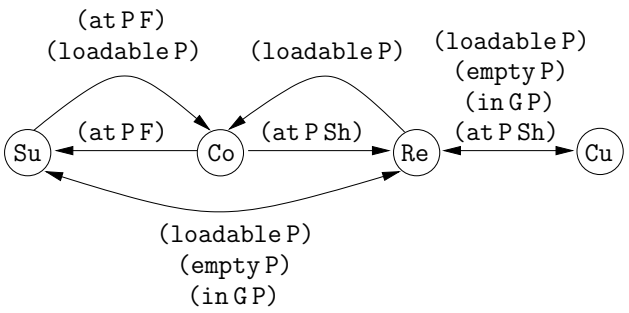


Figure 2: The graph induced by $\{M_i\}_{i=1}^n$ for the multi-agent scenario of Figure 1.

$$rm{-}_i^j(t) = \Big\{p \mid \langle \neg p, \alpha_i \rangle \in \bigcup_k sm{-}_{k \to i}^j(t)\Big\},$$

$$rm{+}_i^j(0) = rm{-}_i^j(0) = \emptyset.$$

Intuitively, for planning step j, $sm{+}_{i\to k}^j(t)/sm{-}_{i\to k}^j(t)$ is the set of positive/negative literals that at the t-th forwarding step ($t = 1 \ldots n-1$) agent α_i sends to agent α_k; $rm{+}_i^j(t)/rm{-}_i^j(t)$ is the set of positive/negative literals that at the t-th forwarding step agent α_i receives. Note that propositional planning assumes that at every planning step the execution of actions is instantaneous, and hence the information exchanges also happens instantaneously.

We say that the single-agent plan π_i is *consistent* if the following conditions hold for $j = 1 \ldots l$ and $t = 1 \ldots n-1$:

(1) $E{+}_i^j = \bigcup_t rm{+}_i^j(t)$, $E{-}_i^j = \bigcup_t rm{-}_i^j(t)$;

(2) $\forall a, b \in A_i^j \cdot Pre(a) \cap Eff{-}(b) = Pre(b) \cap Eff{-}(a) = \emptyset$;

(3) $\forall a, b \in A_i^j \cdot Eff{+}(a) \cap Eff{-}(b) = Eff{+}(b) \cap Eff{-}(a) = \emptyset$;

(4) $\forall a \in A_i^j, \forall e \in E{-}_i^j \cdot Pre(a) \cap e = \emptyset = Eff{+}(a) \cap e = \emptyset$.

Basically, (1) asserts that at planning step j all the exogenous events for agent α_i are the positive/negative literals α_i receives during the information exchange, i.e., (1) guarantees that these events are generated by some other agent; (2) and (3) assert that at planning step j agent α_i executes no pair of mutually exclusive actions; finally, (4) asserts that at planning step j agent α_i executes no action that is mutex with some action executed by other agents.

Let $\langle s_i^0, s_i^1, \ldots, s_i^l \rangle$ be the state trajectory generated by single-agent plan π_i. Plan π_i is executable if $Pre(a) \subseteq s_i^{j-1}$, $\forall a \in A_i^j, j = 1 \ldots l$. Plan π_i is valid for agent α_i if it is executable, consistent, and achieves the goals of agent α_i, i.e., $G_i \subseteq s_i^l$. A multi-agent plan $\{\pi_i\}_{i=1}^n$ is a solution of the multi-agent privacy-preserving planning task if single-agent plan π_i is valid for agent α_i, for $i = 1 \ldots n$.

The main difference with existing models to multi-agent planning, like (Torreño, Onaindia, & Sapena 2012), is related to sets $\{M_i\}_{i=1}^n$ and the purpose for which agents use them. Essentially, M_i determines the messages agent α_i can generate during the execution of its plan, that can be sent to other agents without loss of privacy. Figure 2 shows an example of the graph induced by $\{M_i\}_{i=1}^n$ for the multi-agent scenario of Figure 1. E.g., $\langle (\texttt{in G P}), \texttt{Re} \rangle \in M_{\texttt{Su}}$, and $\langle (\texttt{in G P}), \texttt{Cu} \rangle \in M_{\texttt{Re}}$. Therefore, when agent supplier `Su` packs goods `G` into package `P`, `Su` communicates that `G` is in `P` to agent retailer `Re`; when `Re` receives this communication, `Re` sends it to agent customer `Cu`, so that courier `Co` has no access to message (`in G P`), and there is no direct contact between the retailer's seller and the retailer's customer. A solution multi-agent plan is $\{\pi_{\texttt{Su}}, \pi_{\texttt{Co}}, \pi_{\texttt{Re}}, \pi_{\texttt{Cu}}\}$ with:

$$\pi_{\texttt{Su}} = \langle ((\texttt{pack G P F}), \emptyset), (\emptyset, \{\neg(\texttt{at P F})\}), (\emptyset, \emptyset), (\emptyset, \emptyset), (\emptyset, \{\neg(\texttt{loadable P}), (\texttt{empty P}), \neg(\texttt{in G P})\}) \rangle$$

$$\pi_{\texttt{Co}} = \langle (\emptyset, \{(\texttt{loadable P})\}), ((\texttt{load T P F}), \emptyset), ((\texttt{move T F Sh}), \emptyset), ((\texttt{unload T P Sh}), \emptyset), (\emptyset, \{\neg(\texttt{loadableP})\}) \rangle$$

$$\pi_{\texttt{Re}} = \langle (\emptyset, \{(\texttt{loadable P}), \neg(\texttt{empty P}), (\texttt{in G P})\}) (\emptyset, \emptyset), (\emptyset, \emptyset), (\emptyset, \{(\texttt{at P Sh})\}), (\emptyset, \{\neg(\texttt{loadable P}), (\texttt{empty P}), \neg(\texttt{in G P})\}) \rangle$$

$$\pi_{\texttt{Cu}} = \langle (\emptyset, \{(\texttt{loadable P}), \neg(\texttt{empty P}), (\texttt{in G P})\}) (\emptyset, \emptyset), (\emptyset, \emptyset), (\emptyset, \{(\texttt{at P Sh})\}), ((\texttt{unpack G P Sh}), \emptyset). \rangle$$

In the rest of the paper, we describe how MA-A^* (Nissim, & Brafman 2012) can be adapted to handle our problem model. Briefly, in MA-A^* each agent considers a separate search space, since each agent maintains its own open list and, when an agent expands a state s from its open list, the agent uses its own actions. The open search states that are relevant to different agents are shared, i.e., when s is expanded each agent sends to the others a representation of s obtained by encrypting private propositions.

In order to preserve the privacy according to $\{M_i\}_{i=1}^n$, each agent α_i generates its own key that will be used to encrypt every proposition in F_i except those that α_i sends to or receives from other agents. The agents that are capable of communicating proposition p initially exchange a (shared) key to encrypt p.

At the beginning, each agent α_i constructs its own (partially encrypted) description I_i' of the initial global state of the MA scenario, and α_i sets I_i' to I_i. For each agent α_k α_i is capable to communicate with, α_i encrypts the portion of I_i' formed by all propositions $p \in F_i$ such that $\langle p, \alpha_k \rangle \notin M_i$. Specifically, α_i encrypts p by using the encryption key of p, if it exists; while α_i encrypts p by using its own encryption key, otherwise. Then, α_i sends the resulting state to α_k. When agent α_i

receives a description of the initial state, α_i decrypts the portion of the state formed by the (encrypted) propositions in F_i and computes the union between the resulting state and I'_i. If such a state I''_i is different from I'_i, agent α_i sets I'_i to I''_i and, for each agent α_k α_i is capable to communicate with, α_i sends the description of I'_i to α_k as described before. This procedure is repeated until, for every agent α_i, I'_i does not change anymore. Similarly, subsequently each agent constructs its own (partially encrypted) description G'_i of the initial global set of goals of the MA scenario.

Then, each agents α_i performs the MA-A^* procedure from initial state I'_i to achieve the goals G'_i. The important difference w.r.t. the procedure described in (Nissim, & Brafman 2012) concerns the information exchange among agents. The exchanged messages still include (partially encrypted) description of the world state, but the encrypted propositions of these messages are different. Specifically, when agent α_i expands a state, for every other agent α_k α_i can communicate with, agent α_i encrypts the portion of the state formed by every proposition p such that $\langle p, \alpha_k \rangle \notin M_i$ by using the encryption key of p, if exists, or its own encryption key, otherwise. Then, α_i sends the resulting state to α_k. Note that α_i communicates not only proposition p to agent α_k: agents exchange the complete representation of the search state that can be partially encrypted.

After agent α_i extracts the best state s from open list and generates the successors of s by applying its own actions that are executable from s, α_i has to calculate the h-value of every generated successor state. The heuristic value computed using only the actions of agent α_i can often be inaccurate. In a MA scenario, accurate heuristics can be computed in a distributed way. The drawback of this method is that the computation of the heuristic value can require several information exchanges, and becomes a bottleneck of the search procedure (Stolba, & Komenda 2013). An alternative way for computing accurate heuristics is using a set of actions encrypted according to sets $\{M_i\}_{i=1}^n$. Basically, initially, each agent α_i sends an encrypted representation of its own actions to every agent α_k α_i can communicate with. Specifically, for every action a of α_i, α_i encrypts the name of a using its own encryption key, encrypts every precondition/effect p of a such that $\langle p, \alpha_k \rangle \notin M_i$ by using the encryption key of p, if exists, or its own encryption key, otherwise. This information exchange has the potential disadvantage that an agent could infer some private information from the encrypted representation of the actions of other agents. Whether and how this inference, which may depend on the particular application domain and planning problem, is possible deserves further investigation.

Conclusion

In this paper, we have presented a model of the multi-agent planning task that preserves the agents' privacy, and we have briefly described how the MA-A^* procedure can be adapted to implement the proposed model. Ongoing work includes the implementation of our version of MA-A^*, the study of new heuristics and other algorithms for planning with our model.

Appendix: Encoding of a MA privacy preserving planning task

In this appendix, we show the PDDL encoding of our running example. The agent communication constraints for their information exchange are specified according to the following BNF grammar:

```
<agent-def> ::= (:agent <agent>)
<comm-def> ::= (:communications <comm-list>)
<comm-list> ::= (to <agent> <atomic formula>)
<comm-list> ::= (to <agent> <atomic formula>) <comm-list>
<agent> ::= <name>
<atomic formula> ::= (<predicate> <name>*)
```

Supplier

```
(define (domain example)
(:requirements :typing)
(:agent supplier)
(:types location agent locatable
        goods package - locatable)
(:predicates
        (at ?o - locatable ?l - location)
        (in ?g - goods ?p - package)
        (empty ?p - package)
        (loadable ?p - package))
(:action PACK
 :parameters (?g - goods ?p - package ?l - location)
 :precondition (and (at ?p ?l) (at ?g ?l) (empty ?p))
 :effect (and (not (at ?g ?l)) (in ?g ?p)
              (not (empty ?p)) (loadable ?p))))

(define (problem business1)
(:domain Business)
(:agent Supplier)
(:objects
       Supplier Retailer - agent
       G - goods
       P - package
       Factory - location)
(:init (at P Factory) (at G Factory) (empty P))
(:communications
   (to Retailer (in G P))
   (to Retailer (empty P))
   (to Retailer (loadable P))
   (to Courier (loadable P))
   (to Courier (at P F))))
```

Courier

```
(define (domain example)
  (:requirements :typing)
  (:types location agent locatable
          goods package truck - locatable)
(:agent Courier)
(:predicates
     (at ?o - locatable ?l - location)
     (loadable ?p - package)
     (in ?p - package ?t - truck))
(:action LOAD
  :parameters (?t - truck ?p - package ?l - location)
```

```
  :precondition (and (at ?t ?l) (at ?p ?l)
                     (loadable ?p))
  :effect (and (not (at ?p ?l)) (in ?p ?t)))
(:action DRIVE
  :parameters (?t - truck ?l1 - location ?l2 - location)
  :precondition (and (at ?t ?l1))
  :effect (and (not (at ?t ?l1)) (at ?t ?l2)))
(:action UNLOAD
  :parameters (?t - truck ?p - package ?l - location)
  :precondition (and (in ?p ?t) (at ?t ?l))
  :effect (and (not (in ?p ?t)) (at ?p ?l))) )

(define (problem business1)
(:domain Business)
(:agent Courier)
(:objects
      Supplier Retailer - agent
      P - package
      Factory Shop - location
      T - truck)
(:init (at P Factory) (at T Factory))
(:communications
   (to Retailer (at P Shop))
   (to Supplier (at P Factory))))
```

Customer

```
(define (domain example)
  (:requirements :typing)
  (:agent customer)
  (:types location agent locatable
          goods package - locatable)
  (:predicates
        (at ?o - locatable ?l - location)
        (in ?g - goods ?p - package)
        (empty ?p - package)
        (loadable ?p - package))
  (:action UNPACK
  :parameters (?g - goods ?p - package ?l - location)
  :precondition (and (at ?p ?l) (in ?g ?p))
  :effect (and (not (in ?g ?p)) (at ?g ?l)
               (not (loadable ?p)) (empty ?p))))

(define (problem business1)
(:domain Business)
(:agent Customer)
(:objects
      Retailer - agent
      G - goods
      P - package
      Shop - location)
(:goal (and (at G Shop)))
(:communications
 (to Retailer (loadable P))
 (to Retailer (empty P))
 (to Retailer (in G P))))
```

Retailer

```
(define (domain example)
(:requirements :typing)
(:agent Retailer)
(:types location agent locatable
        goods package - locatable)
(:predicates
        (at ?p - package ?l - location)
        (in ?g - goods ?p - package)
        (empty ?p - package)
        (loadable ?p - package))

(define (problem business1)
(:domain Business)
(:agent Retailer)
(:objects
      Supplier Courier Customer - agent
      P - package
      G - goods
      Shop - location)
(:communications
   (to Customer (in G P))
   (to Customer (empty P))
   (to Customer (loadable P))
   (to Supplier (in G P))
   (to Supplier (empty P))
   (to Supplier (loadable P))
   (to Courier (loadable P))))
```

References

Ronen I. Brafman and Carmel Domshlak, 'From one to many: Planning for loosely coupled multi-agent systems', in *Proc. of the 18th ICAPS*, (2008).

Raz Nissim and Ronen I. Brafman, 'Multi-agent A* for parallel and distributed systems', in *Proc. of the 11th AAMAS*, (2012).

Michal Stolba and Antonín Komenda, 'Fast-Forward Heuristic for Multiagent Planning', in *Proc. of the ICAPS-13 Workshop on Distributed and Multi-agent Planning*, (2013).

Alejandro Torreño, Eva Onaindia, and Óscar Sapena, 'An approach to multi-agent planning with incomplete information', in *Proc. of the 20th ECAI*, (2012).

A Formal Analysis of Required Cooperation in Multi-agent Planning

Yu Zhang and Subbarao Kambhampati
School of Computing and Informatics
Arizona State University
Tempe, Arizona 85281 USA
{yzhan442,rao}@asu.edu

Abstract

Research on multi-agent planning has been popular in recent years. While previous research has been motivated by the understanding that, through cooperation, multi-agent systems can achieve tasks that are unachievable by single-agent systems, there are no formal characteristics of situations where cooperation is required to achieve a goal, thus warranting the application of multi-agent systems. In this paper, we provide such a formal discussion from the planning aspect. We first show that determining whether there is required cooperation (RC) is intractable is general. Then, by dividing the problems that require cooperation (referred to as RC problems) into two classes – problems with heterogeneous and homogeneous agents, we aim to identify all the conditions that can cause RC in these two classes. We establish that when none of these identified conditions hold, the problem is single-agent solvable. Furthermore, with a few assumptions, we provide an upper bound on the minimum number of agents required for RC problems with homogeneous agents. This study not only provides new insights into multi-agent planning, but also has many applications. For example, in human-robot teaming, when a robot cannot achieve a task, it may be due to RC. In such cases, the human teammate should be informed and, consequently, coordinate with other available robots for a solution.

Introduction

A multi-agent planning (MAP) problem differs from a single agent planning (SAP) problem in that more than one agent is used in planning. While a (non-temporal) MAP problem can be compiled into a SAP problem by considering agents as resources, the search space grows exponentially with the number of such resources. Given that a SAP problem with a single such resource is in general PSPACE-complete (Bylander 1991), running a single planner to solve MAP is inefficient. Hence, previous research has generally agreed that agents should be considered as separate entities for planning, and thus has been mainly concentrated on how to explore the interactions between the agents (i.e., loosely-coupled vs. tightly-coupled) to reduce the search space, and how to perform the search more efficiently in a distributed fashion.

However, there has been little discussion on whether multiple agents are required for a planning problem in the first place. If a single agent is sufficient, solving the problem with multiple agents becomes an efficiency matter, e.g., shortening the makespan of the plan. Problems of this nature can be solved in two separate steps: planning with a single agent and optimizing with multiple agents. In such a way, the difficulty of finding a solution may potentially be reduced.

In this paper, we aim to answer the following questions: 1) Given a problem with a set of agents, what are the conditions that make cooperation between multiple agents *required* to solve the problem; 2) How to determine the minimum number of agents required for the problem. We show that providing the exact answers is intractable. Instead, we attempt to provide approximate answers. To facilitate our analysis, we first divide MAP problems into two classes – MAP problems with heterogeneous agents, and MAP problems with homogeneous agents. Consequently, the MAP problems that *require cooperation* (referred to as RC problems) are also divided into two classes – type-1 RC (RC with heterogeneous agents) and type-2 RC (RC with homogeneous agents) problems. Figure 1 shows these divisions.

For the two classes of RC problems, we aim to identify all the conditions that can cause RC. Figure 2 presents these conditions and their relationships to the two classes of RC problems. We establish that at least one of these conditions must be present in order to have RC. Furthermore, we show that most of the problems in common planning domains belong to type-1 RC, which is identified by three conditions in the problem formulation that define the heterogeneity of agents; most of the problems in type-1 RC can be solved by a *super agent*. For type-2 RC, we show that RC is only caused when the state space is not *traversable* or when there are *causal loops* in the causal graph. We provide upper bounds for the answer of the second question for type-2 RC problems, based on different relaxations of the conditions that cause RC, which are associated with, for example, how certain causal loops can be broken in the causal graph.

The answers to these questions not only enrich our fundamental understanding of MAP, but also have many applications. For example, in a human robot teaming scenario, a human may be remotely working with multiple robots. When a robot is assigned a task that it cannot achieve, it is useful to determine whether the failure is due to the fact that the

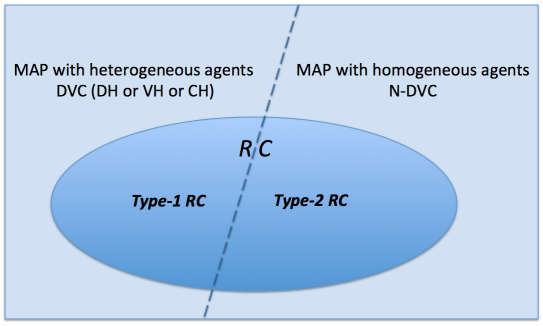


Figure 1: Division of MAP problems into MAP with heterogeneous and homogeneous agents. Consequently, RC problems are also divided into two classes: type-1 RC involves RC problems with heterogeneous agents and type-2 RC involves RC problems with homogeneous agents.

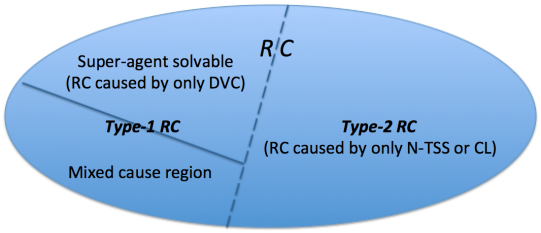


Figure 2: Causes of required cooperation in RC problems.

task is simply unachievable or the task requires more than one robot. In the latter case, it is useful then to determine how many extra robots must be sent to help. The answers can also be applied to multi-robot systems, and are useful in general to any multi-agent systems in which the team compositions can dynamically change (e.g., when the team must be divided to solve different problems).

The rest of the paper is organized as follows. After a review of the related literature, we start the discussion of required cooperation for MAP, in which we answer the above questions in an orderly fashion. We conclude afterward.

Related Work

One of the earlier works on MAP is the PGP framework by (Durfee and Lesser 1991; Decker and Lesser 1992). Recently, the MAP problem has started to receive an increasing amount of attention. Most of these recent research works consider agents separately for planning, and have been concentrated on how to explore the structure of agent interactions to reduce the search space, as well as solving the problem in a distributed fashion. (Nissim, Brafman, and Domshlak 2010) provide a search method by compiling MAP into a constraint satisfaction problem (CSP), and then using a distributed CSP framework to solve it. The MAP formulation is based on an extension of the STRIPS language called MA-STRIPS (Brafman and Domshlak 2008). In MA-STRIPS, actions are categorized into public and private actions. Public actions can influence other agents while private actions cannot. In this way, it is shown by (Brafman and Domshlak 2008) that the search complexity of MAP is exponential in the tree-width of the agent interaction graph. Due to the poor performance of DisCSP based approaches, (Nissim and Brafman 2012) apply the A^* search algorithm in a distributed manner, which represents one of the state-of-art MAP solvers. (Torreno, Onaindia, and Sapena 2012) propose a POP-based distributed planning framework for MAP, which uses a cooperative refinement planning technique that can handle planning with any level of coupling between the agents. Each agent at any step proposes a refinement step to improve the current group plan. Their approach does not assume complete information. A similar paradigm is taken by (Kvarnstrom 2011). An iterative best-response planning and plan improvement technique using standard SAP algorithms is provided by (Jonsson and Rovatsos 2011), which considers the previous singe agent plans as constraints to be satisfied while the following agents perform planning.

Given a problem, all of these MAP approaches solve it using the given set of agents, without first asking whether multiple agents are really required, let alone what is the minimum number of agents required. Answers to these questions not only separate MAP from SAP in a fundamental way, but also have real world applications when the team compositions can dynamically change. In this paper, we analyze these questions using the SAS$^+$ formalism (Backstrom and Nebel 1996) with *causal graph* (Knoblock 1994; Helmert 2006), which is often discussed in the context of factored planning (Bacchus and Yang 1993; Amir and Engelhardt 2003; Brafman 2006; Brafman and Domshlak 2013). The causal graph captures the interaction between different variables; intuitively, it can also capture the interactions between agents since agents affect each other through these variables. In fact, (Brafman and Domshlak 2013) mention the causal graph's relation to the agent interaction graph when each variable is associated with a single agent.

Multi-agent Planning (MAP)

In this paper, we start the analysis of RC in the simplest scenarios – with instantaneous actions and sequential execution. The possibility of RC can only increase when we extend the model to the temporal domain, in which concurrent or synchronous actions must be considered. We develop our analysis of required cooperation for MAP based on the SAS$^+$ formalism (Backstrom and Nebel 1996).

Background

Definition 1. *A SAS$^+$ problem is given by a tuple $P = \langle V, A, I, G \rangle$, where:*

- *$V = \{v_1, ..., v_n\}$ is a set of state variables. Each variable $v_i \in V$ is associated with its domain $D(v_i)$, which is used to define an extended domain $D(v_i)^+ = D(v_i) \cup u$, where u denotes the undefined value. The state space is defined as $S_V^+ = D(v_1)^+ \times ... \times D(v_n)^+$; $s[v_i]$ denotes the value of the variable v_i in a state $s \in S_V^+$.*
- *$A = \{a_1, ..., a_m\}$ is a finite set of actions. Each action a_j is a tuple $\langle pre(a_j), post(a_j), prv(a_j) \rangle$, where $pre(a_j), post(a_j), prv(a_j) \subseteq S_V^+$ are the preconditions, postconditions and prevail conditions of a_j, respectively.*

We also use $pre(a_j)[v_i], post(a_j)[v_i], prv(a_j)[v_i]$ to denote the corresponding values of v_i.

- *I and G denote the initial and goal state, respectively.*

A plan in SAS$^+$ is often defined to be a total-order plan:

Definition 2. *A plan π in SAS$^+$ is a sequence of actions $\pi = \langle a_1, ..., a_l \rangle$.*

Given two states $s_1, s_2 \in S_V^+$, $(s_1 \oplus s_2)$ denotes that s_1 is updated by s_2, and is subject to the following for all $v_i \in V$:

$$(s_1 \oplus s_2)[v_i] = \begin{cases} s_2[v_i] & \text{if } s_2[v_i] \neq u, \\ s_1[v_i] & \text{otherwise.} \end{cases} \quad (1)$$

Given a variable with two values x, y in which one of them is u, $x \sqcup y$ is defined to be the other value. $\sqcup$ can be extended to two states s_1 and s_2, such that $s_1 \sqcup s_2[vi] = s_1[v_i] \sqcup s_2[v_i]$ for all $v_i \in V$. $s_1 \sqsubseteq s_2$ if and only if $\forall v_i \in V, s_1[v_i] = u$ or $s_1[v_i] = s_2[v_i]$. The state resulting from executing a plan π can then be defined recursively using a re operator as follows:

$$re(s, \langle \pi; o \rangle) = \begin{cases} re(s, \langle \pi \rangle) \oplus post(o) \\ \text{if } pre(o) \sqcup prv(o) \sqsubseteq re(s, \langle \pi \rangle), \\ s \quad \text{otherwise.} \end{cases} \quad (2)$$

in which $re(s, \langle \rangle) = s$, o is an action, and ; is the concatenation operator.

Extension to MAP

To extend the previous formalism to MAP without losing generality, we minimally modify the definitions.

Definition 3. *A SAS$^+$ MAP problem is given by a tuple $\Pi = \langle V, \Phi, I, G \rangle$, where:*

- *$\Phi = \{\phi_g\}$ is the set of agents; each agent ϕ_g is associated with a set of actions $A(\phi_g)$.*

Definition 4. *A plan π_{MAP} in MAP is a sequence of agent-action pairs $\pi_{MAP} = \langle (a_1, \phi(a_1)), ..., (a_L, \phi(a_L)) \rangle$, in which $\phi(a)$ returns the agent for the action a and L is the length of the plan.*

We do not need to consider concurrency or synchronization given that actions are assumed to be instantaneous.

Required Cooperation for MAP

Next, we formally define the notion of *required cooperation* and other useful terms that are used in the following analyses. We assume throughout the paper that more than one agent is considered (i.e., $|\Phi| > 1$).

Required Cooperation

Definition 5 (k-agent Solvable)**.** *Given a MAP problem $P = \langle V, \Phi, I, G \rangle$ ($|\Phi| \geq k$), the problem is k-agent solvable if $\exists \Phi_k \subseteq \Phi$ ($|\Phi_k| = k$), such that $\langle V, \Phi_k, I, G \rangle$ is solvable.*

Definition 6 (Required Cooperation (RC))**.** *Given a solvable MAP problem $P = \langle V, \Phi, I, G \rangle$, there is required cooperation if it is not 1-agent solvable.*

In other words, given a solvable MAP problem that satisfies RC, any plan must involve more than one agent.

Lemma 1. *Given a solvable MAP problem $P = \langle V, \Phi, I, G \rangle$, determining whether it satisfies RC is PSPACE-complete.*

Proof. First, it is not difficult to show that the RC decision problem belongs to PSPACE, since we only need to verify that $P = \langle V, \phi, I, G \rangle$ is unsolvable for all $\phi \in \Phi$, given that the initial problem is known to be solvable. Then, we complete the proof by reducing from the PLANSAT problem, which is PSPACE-complete in general (Bylander 1991). Given a PLANSAT problem (with a single agent), the idea is that we can introduce a second agent with only one action. This action directly achieves the goal but requires an action (with all preconditions satisfied in the initial state) of the initial agent to provide a precondition that is not initially satisfied. We know that this constructed MAP problem is solvable. If the algorithm for the RC decision problem returns that cooperation is required for this MAP problem, we know that the original PLANSAT problem is unsolvable; otherwise, it is solvable. □

Definition 7 (Minimally k-agent Solvable)**.** *Given a solvable MAP problem $P = \langle V, \Phi, I, G \rangle$ ($|\Phi| \geq k$), it is minimally k-agent solvable if it is k-agent solvable, and not $(k-1)$-agent solvable.*

Corollary 1. *Given a solvable MAP problem $P = \langle V, \Phi, I, G \rangle$, determining the minimally solvable k ($k \leq |\Phi|$) is PSPACE-complete.*

Although directly querying for RC is intractable, we aim to identify all the conditions (which can be quickly checked) that can cause RC. We first define a few terms that are used in the following discussions.

We note that the reference of agent is explicit in the action (i.e., ground operator) parameters. Although actions are unique for each agent, two different agents may be capable of executing actions that are instantiated from the same operator, with all other parameters being identical. To identify such cases, we introduce the notion of *action signature*.

Definition 8 (Action Signature (AS))**.** *An action signature is an action with the reference of the executing agent replaced by a global $EX\text{-}AG$ symbol.*

For example, an action signature in the IPC logistics domain is $drive(EX\text{-}AG, pgh\text{-}po, pgh\text{-}airport)$. $EX\text{-}AG$ is a global symbol to denote the executing agent, which is not used to distinguish between action signatures. We denote the set of action signatures for $\phi \in \Phi$ as $AS(\phi)$, which specifies the *capabilities* of ϕ. Furthermore, we define the notion of *agent variable*.

Definition 9 (Agent Variable (Agent Fluent))**.** *A variable (fluent) is an agent variable (fluent) if it is associated with the reference of an agent.*

Agent variables are used to specify agent state. For example, $location(truck\text{-}pgh)$ is an agent variable since it is associated with an agent $truck\text{-}pgh$. We use $V_\phi \subseteq V$ to denote the set of agent variables that are associated with ϕ (i.e., variables that are present in the initial state or actions of ϕ).

Following this notation, we can rewrite a MAP problem as $P = \langle V_o \cup V_\Phi, \Phi, I_o \cup I_\Phi, G_o \cup G_\Phi \rangle$, in which $V_\Phi = \{V_\phi\}$,

$I_\Phi = \{I_\phi\}, G_\Phi = \{G_\phi\}, I_\phi = I \cap V_\phi$ and $G_\phi = G \cap V_\phi$. V_o denotes the set of non-agent variables; I_o and G_o are the set of non-agent variables in I and G, respectively. In this paper, we assume that agents can only interact with each other through non-agent variables (i.e., V_o). In other words, agent variables contain one and only one reference of agent. As a result, we have $V_\phi \cap V_{\phi'} \equiv \emptyset$ ($\phi \neq \phi'$). It seems to be possible to compile away exceptions by breaking agent variables (with more than one reference of agent) into multiple variables and introducing non-agent variables to correlate them.

Definition 10 (Variable (Fluent) Signature (VS))**.** *Given an agent variable (fluent), its variable (fluent) signature is the variable (fluent) with the reference of agent replaced by* $EX\text{-}AG$.

For example, $location(truck\text{-}pgh)$ is an agent variable for $truck\text{-}pgh$ and its variable signature is $location(EX\text{-}AG)$. We denote the set of VSs for V_ϕ as $VS(\phi)$, and use VS as an operator so that $VS(v)$ returns the VS of a variable v; this operator returns any non-agent variable unchanged.

Classes of RC

In the following discussion, we assume that the specification of goal (i.e., G) in the MAP problems does not involve agent variables (i.e., $G \cap V_\phi = \emptyset$ or $G_\phi = \emptyset$), since we are mostly interested in how to reach the desired world state (i.e., specified in terms of V_o). As aforementioned, we divide RC problems into two classes as shown in Figure 1. Type-1 RC involves problems with heterogeneous agents; type-2 RC involves problems with homogeneous agents. Next, we formally define each class and discuss the causes of RC. Throughout this paper, when we denote a condition as X, the negated condition is denoted as N-X.

Type-1 RC (RC with Heterogeneous Agents)

Given a MAP problem $P = \langle V, \Phi, I, G \rangle$, the heterogeneity of agents can be characterized by the following conditions:

- *Domain Heterogeneity (DH)*: $\exists v \in V_\phi$ and $D(v) \setminus D(V') \neq \emptyset$, in which $V' = \{v' | v' \in V_{\phi'} (\phi' \neq \phi)$ and $VS(v) = VS(v')\}$.
- *Variable Heterogeneity (VH)*: $VS(\phi) \setminus VS(\Phi \setminus \phi) \neq \emptyset$.
- *Capability Heterogeneity (CH)*: $AS(\phi) \setminus AS(\Phi \setminus \phi) \neq \emptyset$.

Definition 11 (Type-1 RC)**.** *An RC problem belongs to type-1 RC if at least one of DH, VH and CH is satisfied for an agent.*

The condition that requires at least one of DH, VH and CH to be satisfied is denoted as DVC in Figure 1. It is worth noting that when considering certain objects (e.g., truck and plane in the logistics domain) as agents rather than as resources, most of the RC problems in the IPC domains belong to type-1 RC.

Causes of RC in Type-1

The most obvious condition for RC in type-1 RC problems is due to the heterogeneity of agents. In the logistics domain, for example, if any truck agent can only stay in one city, the domains of the location variable for different truck agents are different (DH). When there are packages that must be transferred between different locations within cities, at least one truck agent for each city is required (hence RC). In the rover domain, a rover that is not equipped with a camera sensor would not be associated with the agent variable $equipped\_for\_imaging$. When we need both $equipped\_for\_imaging$ and $equipped\_for\_rock\_analysis$, and no rovers are equipped with the sensors for both (VH), we have RC. Note that VH does not specify any requirement on the variable value (i.e., the state); however, when the domain of a variable contains only a single value, e.g., $equipped\_for\_imaging$, we assume in this paper that this variable is always defined in a positive manner, e.g., expressing cans instead of cannots. In the logistics domain, given that the truck agent cannot fly (CH), when a package must be delivered from a city to a non-airport location of another city, at least a truck and a plane are required. Note that DH, VH and CH are closely correlated.

However, note that 1) the presence of DVC in a solvable MAP problem does not always cause RC, as shown in Figure 1; 2) the presence of DVC in a type-1 RC problem is not always the cause of RC, as shown in Figure 2.

As an example for 1), when there is only one package to be delivered from one location to another within the same city, there is no need for a plane agent, even though we can create a non-RC MAP problem with a plane and a truck agent that satisfies CH (thus DVC).

As an example for 2), for navigating in a grid world, the traversability of the world for all mobile agents can be restricted based on edge connections, i.e., $connected(a, b)$, in which a and b are vertices in the grid. Suppose that we have two packages to be delivered to locations b and c, respectively, which are both initially at a. There are two truck agents at a that can be used for delivery. However, the paths from a to both b and c are one-way only (i.e., $connected(a, b) = true$ and $connected(b, a) = false$). Even if one of the truck agents uses gas and the other one uses diesel, thus satisfying DVC, it is clear that RC in this problem is not caused by the heterogeneity of agents.

Type-1 RC problems in which RC is caused by only DVC can be solved by a *super agent* (defined below), which is an agent that combines all the domain values, variable signatures and capabilities (i.e., action signatures) of the other agents. We refer to the subset of type-1 RC problems that can be solved by a super agent as *super-agent solvable*, as shown in Figure 2.

Definition 12 (Super Agent)**.** *A super agent is an agent* ϕ^* *that satisfies:*

- $\forall v \in V_\Phi$, $\exists v^* \in V_{\phi^*}$, $D(v^*) = D(V)$, *in which* $V = \{v | v \in V_\Phi$ *and* $VS(v^*) = VS(v)\}$.
- $VS(\phi^*) = VS(\Phi)$.
- $AS(\phi^*) = AS(\Phi)$.

It is not difficult to see that most problems in the IPC domains are also super-agent solvable. For example, when we have a truck-plane agent in the logistics domain that can both fly (between airports of different cities) and drive (between locations in the same cities), or when we have a rover that is

equipped with all sensors and can traverse all waypoints in the rover domain.

From Figure 2, one may have already noticed that the conditions that cause RC in type-2 problems may also cause RC in type-1 problems (i.e., indicated by the *mixed cause region* in Figure 2). For example, the aforementioned example for navigating in a grid world demonstrates that the initial states (specified in terms of the values for variables) of different agents may cause RC in type-1 problems. Note that the initial states of different agents cannot be combined as for domain values, variable signatures and capabilities in a super agent construction; however, the special cases when the domains of variables contain only a single value (when we discussed VH in **Causes of RC in Type-**1) can also be considered as cases when RC is caused by the initial state.

Type-2 RC (RC with Homogeneous Agents)

Type-2 RC involves homogeneous agents:

Definition 13 (Type-2 RC)**.** *An RC problem belongs to type-2 RC if it satisfies N-DVC (for all agents).*

Definition 13 states that an RC problem belongs to type-2 RC when all the agents are homogeneous.

Type-2 RC Caused by Traversability

One condition that causes RC in type-2 RC problems is the *traversability* of the state space of variables, which is related to the initial states of the agents and the world, as we previously discussed. Since the traversability is associated with the evolution of variable values, we use causal graphs to perform the analysis.

Definition 14 (Causal Graph)**.** *Given a MAP problem $P = \langle V, \Phi, I, G \rangle$, the causal graph G is a graph with directed and undirected edges over the nodes V. For two nodes v and v' ($v \neq v'$), a directed edge $v \rightarrow v'$ is introduced if there exists an action that updates v' while having a prevail condition associated with v. An undirected edge $v - v'$ is introduced if there exists an action that updates both.*

A typical example of a causal graph for an individual agent is presented in Figure 3. For type-2 RC study, since the agents are homogeneous, the causal graphs for all agents are the same. Hence, we can use agent VSs to replace agent variables; we refer to this modified causal graph for a single agent in a type-2 RC problem as an *individual causal graph signature* (ICGS). Next, we define the notions of *closures* and *traversable state space*.

Definition 15 (Inner and Outer Closures (IC and OC))**.** *An inner closure (IC) in an ICGS is any set of variables for which no other variables are connected to them with undirected edges; an outer closure (OC) of an IC is the set of nodes that have directed edges going into nodes in the IC.*

In Figure 3, $\{v_2, v_3\}$ and $\{v_4\}$ are examples of ICs. The OC of $\{v_2, v_3\}$ is $\{v_1\}$ and the OC of $\{v_4\}$ is $\{v_3\}$.

Definition 16 (Traversable State Space (TSS))**.** *An IC has a traversable state space if and only if: given any two states of this IC, denoted by s and s', there exists a plan that connects them, assuming that the state of the OC of this IC can be changed freely within its state space.*

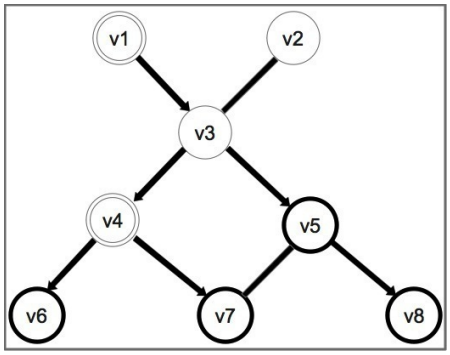


Figure 3: Example of a causal graph (ICGS). Variables in goal G are shown as bold-circle nodes and agent VSs are shown as double-circle nodes.

In other words, an IC has a TSS if the traversal of its state space is only dependent on the variables in its OC; this also means that when the OC of an IC is empty, the state of the IC can change freely. Note that static variables in the OC of an IC can assume values that do not influence the traversability. For example, the variables that are used to specify the connectivity of vertices in a grid, e.g., $connected(a, b)$, can be assigned to be $true$ or $false$; although the variables that are assigned to be $true$ cannot change their values to be $false$, they do not influence the traversability of the grid world. In such cases, the associated ICs are still considered to have a TSS. An ICGS in which all ICs have TSSs is referred to as being traversable.

Type-2 RC Caused by Causal Loops

However, even a solvable MAP problem that satisfies N-DVC for all agents while having a traversable ICGS can still satisfy RC. An example is presented below.

The goal of this problem is to steal a diamond from a room, in which the diamond is secured, and place it in another room. The diamond is protected by a stealth detection system. If the diamond is taken, the system locks the door of the room in which the diamond is kept, so that the insiders cannot exit. There is a switch to override the detection system but it is located outside of the room. This problem is modeled as above, in which the value is immediately specified after each variable. It is not difficult to see that the above problem cannot be solved with a single agent.

Initial State:

$location(agent1)\ room1$
$location(agent2)\ room1$
$location(diamond1)\ room1$
$doorLocked(room1)\ false$
$location(switch1)\ room2$

Goal State:

$location(diamond1)\ room2$

Operators:
$WalkThrough(agent, door, fromRoom, toRoom)$:
prv: $doorLocked(door)$ $false$
pre: $location(agent)$ $fromRoom$
$post$: $location(agent)$ $toRoom$
$Steal(agent, diamond, room, door)$:
prv: $location(agent)$ $room$
pre: $doorLocked(door)$ u
pre: $location(diamond)$ $room$
$post$: $doorLocked(door)$ $true$
$post$: $location(diamond)$ $agent$
$Switch(agent, switch, room, door)$:
prv: $location(switch)$ $room$
prv: $location(agent)$ $room$
pre: $doorLocked(door)$ u
$post$: $doorLocked(door)$ $false$
$Place(agent, diamond, room)$:
prv: $location(agent)$ $room$
pre: $location(diamond)$ $agent$
$post$: $location(diamond)$ $room$

Again, we construct the ICGS for this type-2 RC example, as shown in Figure 4. One key observation is that a single agent cannot address this problem due to the fact that $WalkThrough$ with the diamond to $room2$ requires $doorLocked(door1) = false$, which is violated by the $Steal$ action to obtain the diamond in the first place. This is clearly related to the loops in Figure 4. In particular, we define the notion of *causal loops*.

Definition 17 (Causal Loop (CL))**.** *A causal loop in the ICGS is a directed loop that contains at least one directed edge.*

Note that undirected edges can be considered as edges in either direction but at least one directed edge must be present in a causal loop.

Gap between MAP and Single Agent Planning

We now establish in the following theorem that when none of the previously discussed conditions (for both type-1 and type-2 RC) hold in a MAP problem, this problem can be solved by a single agent.

Theorem 1. *Given a solvable MAP problem that satisfies N-DVC for all agents, and for which the ICGS is traversable and contains no causal loops, any single agent can also achieve the goal.*

Proof. Given no causal loops, the directed edges in the ICGS divides the variables into levels, in which: 1) variables at each level do not appear in other levels; 2) higher level variables are connected to lower level variables with only directed edges going from higher levels to lower levels; 3) variables within each level are either not connected or connected with undirected edges. For example, the variables in Figure 3 are divided into the following levels (from high to low): $\{v_1\}$, $\{v_2, v_3\}$, $\{v_4\}$, $\{v_5, v_7\}$, $\{v_6, v_8\}$. Note that this division is not unique.

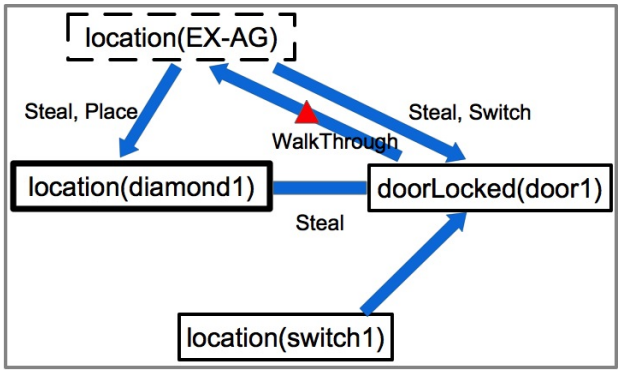


Figure 4: ICGS for the diamond example that illustrates the second condition that causes RC in type-2 RC problems. Actions (without parameters) are labeled along with their corresponding edges. The variables in G are shown as bold-box nodes and agent VSs are shown as dashed-box nodes.

Next, we prove the result by induction based on the level. Suppose that the ICGS has k levels and we have the following holds: given any trajectory of states for all variables, there exists a plan whose execution traces of states include this trajectory in the correct order.

When the ICGS has $k+1$ levels: given any state s for all variables from level 1 to $k+1$, we know from the assumption that the ICGS is traversable that there exists a plan that can update the variables at the $k+1$ level from their current states to the corresponding states in s. This plan (denoted by π), meanwhile, requires the freedom to change the states of variables from level 1 to k. Given the induction assumption, we know that we can update these variables to their required states in the correct order to satisfy π; furthermore, these updates (at level k and above) also do not influence the variables at the $k+1$ level (hence do not influence π). Once the states of the variables at the $k+1$ level are updated to match those in s, we can then update variables at level 1 to k to match their states in s accordingly. Using this process, we can incrementally build a plan whose execution traces of states contain any given trajectory of states for all the variables in the correct order.

Furthermore, the induction holds when there is only one level given that ICGS is traversable. Hence, the induction conclusion holds. The main conclusion directly follows. □

Towards an Upper Bound for Type-2 RC

In this section, we investigate type-2 RC problem to obtain upper bounds on the k (Definition 7), based on different relaxations of the two conditions that cause RC in type-2 RC problems. We first relax the assumption regarding causal loops (CLs) and show that the relaxation process is associated with how certain CLs can be broken.

We notice that there are two kinds of CLs in ICGS. The first kind contains agent VSs while the second kind does not. Although we cannot break CLs for the second kind, it is possible to break CLs for the first kind. The motivation is that certain edges in these CLs can be removed when there is

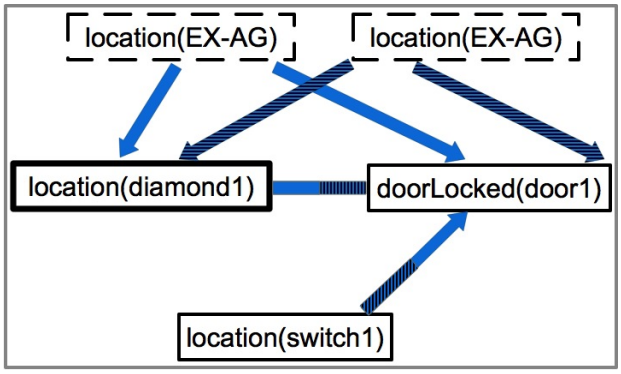


Figure 5: Illustration of the process for breaking causal loops in the diamond example, in which the CLs are broken by removing the edge marked with a triangle in Figure 4. Two agent VSs are introduced to replace the original agent VS.

no need to update the associated agent VSs. In our diamond example, when there are two agents in $room1$ and $room2$, respectively, and they can stay where they are during the execution of the plan, there is no need to $WalkThrough$ and hence the associated edges can be removed to break the CLs. Figure 5 shows this process. Based on this observation, we introduce the following lemma.

Lemma 2. *Given a solvable MAP problem that satisfies N-DVC for all agents and for which the ICGS is traversable, if no CLs contain agent VSs and all the edges going in and out of agent VSs are directed, the minimum number of agents required is upper bounded by $\times_{v \in CR(\Phi)} |D(v)|$, when assuming that the agents can choose their initial states, in which $CR(\Phi)$ is constructed as follows:*

1. *add the set of agent VSs that are in the CLs into $CR(\Phi)$;*
2. *add in an agent VS into $CR(\Phi)$ if there exists a directed edge that goes into it from any variable in $CR(\Phi)$;*
3. *iterate 2 until no agent VSs can be added.*

Proof. Based on the previous discussions, we can remove edges that are connected to agent VSs to break loops. For each variable in $CR(\Phi)$, denoted by v, we introduce a set of variables $N = \{v_1, v_2, ..., v_{|D(v)|}\}$ to replace v. Any edges connecting to v from other variables are duplicated on all variables in N, except for the edges that go into v. Each variable $v_i \in N$ has a domain with a single value; this value for each variable in N is different and chosen from $D(v)$. Note that these new variables do not affect the traversability of the ICGS.

From Theorem 1, we know that a virtual agent ϕ^+ that can simultaneously assume all the states that are the different permutations of states for $CR(\Phi)$ can achieve the goal. We can simulate ϕ^+ using $\times_{v \in CR(\Phi)} |D(v)|$ agents as follows. We choose the agent initial states according to the permutations of states for $CR(\Phi)$, while choosing the same states for all the other agent VSs according to ϕ^+. Given a plan for ϕ^+, we start from the first action. Given that all permutations of states for $CR(\Phi)$ are assumed by an agent, we can find an agent, denoted by ϕ, that can execute this action: 1) If this action updates an agent VS in $CR(\Phi)$, we do not need to execute this action based on the following reasoning. Given that all edges going in and out of agent VSs are directed, we know that this action does not update V_o. (Otherwise, there must be an undirected edge connecting a variable in V_o to this agent VS. Similarly, we also know that this action does not update more than one agent VS.). As a result, it does not influence the execution of the next action. 2) If this action updates an agent VS that is not in $CR(\Phi)$, we know that this action cannot have variables in $CR(\Phi)$ as preconditions or prevail conditions, since otherwise this agent VS would be included in $CR(\Phi)$ given its construction process. Hence, all the agents can execute the action to update this agent VS, given that all the agent VSs outside of $CR(\Phi)$ are always kept synchronized in the entire process (in order to simulate ϕ^+). 3) Otherwise, this action must be updating only V_o and we can execute the action on ϕ.

Following the above process for all the actions in ϕ^+'s plan to achieve the goal. Hence, the conclusion holds. □

Next, we investigate the relaxation of the traversability of the ICGS.

Lemma 3. *Given a solvable MAP problem that satisfies N-DVC for all agents, if all the edges going in and out of agent VSs are directed, the minimum number of agents required is upper bounded by $\times_{v \in VS(\Phi)} |D(v)|$, when assuming that the agents can choose their initial states.*

Proof. Given a valid plan π_{MAP} for the problem, we can solve the problem using $\times_{v \in VS(\Phi)} |D(v)|$ agents as follows: first, we choose the agent initial states according to the permutations of state for $VS(\Phi)$.

The process is similar to that in Lemma 2. We start from the first action. Given that all permutations of $VS(\Phi)$ are assumed by an agent, we can find an agent, denoted by ϕ, that can execute this action: if this action updates some agent VSs in $VS(\Phi)$, we do not need to execute this action; otherwise, the action must be updating only V_o and we can execute the action on ϕ.

Following the above process for all the actions in π_{MAP} to achieve the goal. Hence, the conclusion holds. □

Note that the bounds in Lemma 2 and 3 are upper bounds and the minimum number of agents actually required may be smaller. Nevertheless, for the simple scenario in our diamond example, the assumptions of both lemmas are satisfied and the bounds returned are 2 for both, which happens to be exactly the k in Definition 7. In future work, we plan to investigate other relaxations and establish the tightness of these bounds.

Conclusion

In this paper, we introduce the notion of required cooperation (RC), which answers two questions: 1) whether more than one agent is required for a solvable MAP problem, and 2) what is the minimum number of agents required for the problem. We show that the exact answers to these questions are difficult to provide. To facilitate our analysis, we first divide RC problems into two class – type-1 RC involves

heterogeneous agents and type-2 RC involves homogeneous agents. For the first question, we show that most of the problems in the common planning domains belong to type-1 RC; the set of type-1 RC problems in which RC is only caused by DVC can be solved with a super agent. For type-2 RC problems, we show that RC is caused when the state space is not traversable or when there are causal loops in the causal graph; we provide upper bounds for the answer of the second question, based on different relaxations of the conditions that cause RC in type-2 RC problems. These relaxations are associated with, for example, how certain causal loops can be broken in the causal graph.

Acknowledgement

This research is supported in part by the ARO grant W911NF-13-1-0023, and the ONR grants N00014-13-1-0176 and N00014-13-1-0519.

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In *Proceedings of the 18th International Joint Conferences on Artificial Intelligence*, 929–935.

Bacchus, F., and Yang, Q. 1993. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence* 71:43–100.

Backstrom, C., and Nebel, B. 1996. Complexity results for sas+ planning. *Computational Intelligence* 11:625–655.

Brafman, R. I., and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 28–35. AAAI Press.

Brafman, R. I., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198(0):52 – 71.

Brafman, R. I. 2006. Factored planning: How, when, and when not. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 809–814.

Bylander, T. 1991. Complexity results for planning. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, volume 1, 274–279.

Decker, K. S., and Lesser, V. R. 1992. Generalizing the partial global planning algorithm. *International Journal of Cooperative Information Systems* 1:319–346.

Durfee, E., and Lesser, V. R. 1991. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics* 21:1167–1183.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Jonsson, A., and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *Proceedings of the 21th International Conference on Automated Planning and Scheduling*, 114–121. AAAI Press.

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68:243–302.

Kvarnstrom, J. 2011. Planning for loosely coupled agents using partial order forward-chaining. In *Proceedings of the 21th International Conference on Automated Planning and Scheduling*.

Nissim, R., and Brafman, R. I. 2012. Multi-agent a* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, volume 3, 1265–1266.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 1323–1330.

Torreno, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *European Conference on Artificial Intelligence*, volume 242, 762–767.

Plan Merging by Reuse for Multi-Agent Planning

Nerea Luis and **Daniel Borrajo**
Departamento de Informática
Universidad Carlos III de Madrid
nluis@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract

Multi-Agent Planning (MAP) can be solved by two alternative approaches: centralized, where a single planner computes a plan for multiple agents; and distributed, where each agent computes a plan, and then plans are merged or coordinated. This paper focuses on the second approach. We present Plan Merger by Reuse, PMR, an algorithm that, given a multi-agent planning problem, lets the agents build their individual plans separately. Then, PMR concatenates all the plans to build a parallel plan. As plans are not always valid, specially in the case of tightly-coupled domains, PMR executes LPG-ADAPT with the invalid plan as input. LPG-ADAPT performs planning by reuse and is able to generate a valid plan from the input invalid plan.We show experimental results in several IPC domains, where plan merging can successfully compete against other state-of-the-art planning techniques to obtain solutions to MAP tasks.

Introduction

In Multi-Agent Planning (MAP), two main approaches have been commonly used: centralized and distributed. The former builds a plan using a master agent which knows everything about the other agents. Among other disadvantages, its complexity can grow exponentially with the number of agents. In the distributed approach each agent builds its own plan and then those plans have to be either merged before execution starts, or execution requires some coordination to avoid an agent to negatively interact with other agents goals (e.g. using a common resource).

The goal of plan merging is to start with a set of plans, one for each agent, and then generate a single valid plan. Thus, the resulting plan should be: free of negative interactions among the different agents plans (as one agent deleting some effect that is a precondition of another agent plan); free of actions that achieve goals unnecessarily (as two agents achieving the same goal twice); and sound (when the plan is applied from the overall initial state, it should arrive to a state where goals are achieved). In the past, most plan merging approaches focused on analyzing the input plans and detect positive or negative interactions among the plans (Foulser, Li, and Yang 1992; Mali 2000).

From the perspective of MAP, planning domains exhibit a coupling level that ranges from loosely-coupled to tightly-coupled, depending on the degree of interaction between agents plans (Brafman and Domshlak 2013). This paper describes an approach, Plan Merger by Reuse (PMR), that tries to automatically accommodate to the coupling level of the domain. First, it performs individual planning for each agent and then merges the results into a combined plan and parallelizes it. If the domain is loosely-coupled, then most probably this plan will be a valid plan, since there will be little (or no) interaction among agents plans, and PMR returns that plan. Otherwise (in more tightly-coupled domains), PMR provides that plan as input to a planning by reuse technique (LPG-ADAPT (Fox et al. 2006)) that generates a valid plan from the invalid input plan. The hypothesis is that most of the relevant actions of the final valid plan will be available in the input invalid plan, so LPG-ADAPT will efficiently generate a plan. Experimental results show the benefits of using such an approach, even against a competitive centralized approach (LAMA (Richter and Westphal 2010)).

The next section presents a formal definition of the MAP task. Then, in Section 3, the algorithm PMR and its different phases are described. Section 4 presents the experiments and results of comparing PMR with other approaches. Finally, we present some conclusions and future work to be done.

Multi-Agent Planning Task

A single-agent STRIPS planning task can formally defined as a tuple $\Pi = \{F, A, I, G\}$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of goals. Each action $a \in A$ is described by a set of preconditions (pre(a_i)), that represent literals that must be true in a state to execute the action and a set of effects (eff(a_i)), literals that are expected to be added (add effects) or removed (delete effects) from the state after execution of the action. Actions definition might also include a cost $c(a)$ (default is one). In order to compactly represent planning tasks, automated planning uses the standard language PDDL (Planning Domain Description Language). Thus, a planning task Π is automatically generated from the PDDL description of a domain and a problem. The domain contains a definition of a set of generalized actions (defined using variables – parameters, par(a) – whose instantiations with problem objects will lead to actions in A), a set of pred-

icates (whose instantiations will generate facts in F), and a set of types (to characterize the problem objects). A planning problem defines a set of objects (instantiations of types in the domain), an initial state (I), and a set of goals (G). The planning task should generate as output a sequence of actions $\pi = (a_1, \ldots, a_n)$ such that if applied in order would result in a state s, where goals are true, $G \subseteq s$. Plan cost is commonly defined as: $C(\pi) = \sum_{a_i \in \pi} c(a_i)$.

We consider a multi-agent setting, so we have to plan for a set of m agents, $\Phi = \{\phi_1, \ldots, \phi_m\}$. We define the MAP task as a set of planning subtasks, one for each agent, $M = \{\Pi_1, \ldots, \Pi_m\}$. Each planning subtask can be defined as a single-agent planning task, $\Pi_i = \{A_i, F_i, I_i, G_i\}$. All these components have a public part and a private part, that cannot be openly shared with the rest of agents. We believe privacy relates to the information on the state and the objects in the problems, rather than to the actions. Our notion of privacy takes into account the following statements:

- literals $l \in F$ are considered either private or public. In case they are private, they belong to a given agent a_i and they should only be known and modified by a_i when planning. In case they are needed by other agents to reproduce the agent a_i plans (as explained later), they will have to be obfuscated by a_i when sent to other agents
- in particular, literals $l \in I$ and $l \in G$ can be private or public in turn. If they need to be sent to other agents, they will have to be obfuscated
- actions $a \in A$ are not considered private nor public by themselves. But, we make the assumption that other agents do not need to know the actions that were used to achieve the agents goals (even the public ones). Therefore, when plans are communicated to the central agent in PMR, actions are obfuscated. In particular, the action names and private components (literals) are obfuscated.

As an example, in the Satellite domain of the International Planning Competition (IPC)[1] several satellites must take images from different directions in space. The private literals are those that are groundings of the predicates: supports, calibration_target, on_board, pointing, power_avail, calibrated and power_on.

PMR

The main steps of the PMR algorithm are (as shown in the pseudocode in Figure 1): assign public goals to agents, concatenate each plan built from each agent, parallelize that plan and check if it is a valid plan. $\uplus$ represents the operator that returns the concatenation of individual plans.

Figure 2 shows the architecture of PMR. Giving a multi-agent domain and problem in standard PDDL, two approaches can be chosen: centralized or distributed. We will focus on the distributed, but later we will use the centralized to compare those plans with the ones generated in the distributed approach. First, PMR performs an assignment of goals (public) to agents, as explained later and taken from (Borrajo 2013). After that, for each agent, a total-order plan is generated based on its assigned goals. In this step, each agent will use a planner to generate its plan. For simplicity, in the experiments we assume all agents use the same planner.

[1] http://ipc.icaps-conference.org/

Function PMR (M, GA, P): plan

$M = \{\Pi_1, \ldots, \Pi_m\}$: MAP task
GA: goal assignment strategy
P: planner

Assign subset of public goals to each agent ϕ_i using GA
For all $\phi_i \in \Phi$ do π_i =Plan(Π_i, P)
$\pi_{PMR-seq} = \uplus \pi_i$
π_{par} =parallelize($\pi_{PMR-seq}$)
if valid(π_{par})
then return $\pi_{PMR} = \pi_{par}$
else return $\pi_{PMR-reuse}$ =plan-reuse-planner(π_{par}, M)

Figure 1: High level description of PMR algorithm.

Once all the plans are generated, they are concatenated to create a new plan ($\pi_{PMR-seq}$). Then, PMR transforms $\pi_{PMR-seq}$ into a parallel plan (π_{par}) in order to obtain a better plan in terms of concurrency. This is performed in two steps: converting a total-order plan into a partial-order one by a similar algorithm to (Veloso, Pérez, and Carbonell 1990); and parallelizing this partial-order plan. $\pi_{PMR-seq}$ and π_{par} are validated using VAL, the validator from IPC 2011 (Howey, Long, and Fox 2004). In case, the parallel plan is valid (usually in domains that are loosely coupled as shown in the experiments), PMR returns that plan. Otherwise, PMR executes a plan reuse planner, and π_{par} becomes its input plan. The goal of this step is to find a new plan, by combining the reuse of actions of π_{par}, that is partially valid, and the search heuristics of the plan reuse planner.

Goal Assignment

In a MAP task, the way the public goals are assigned to the different agents affects directly to the efficiency and the performance of the plan. We use here the four goal assignment (GA) strategies defined for MAPR (Borrajo 2013). First, for each agent $\phi_i \in \Phi$ and goal $g \in G$, a relaxed plan is computed (Hoffmann and Nebel 2001). The main objective is to know if the goal can be reached from the initial state of the agent and with what estimated cost. All values fill a cost matrix $c(G, \Phi)$ where each cell represents the estimated cost per agent to achieve a goal. The GA strategies were:

- all-achievable: it assigns each goal to every agent that can reach the goal
- rest-achievable: it assigns the goals iteratively, starting with ϕ_1. First, it assigns all goals that ϕ_1 can potentially achieve to ϕ_1. Those goals are deleted from G. Then, it starts with the second agent and so on.
- best-cost: each goal is assigned to the agent that can potentially achieve it with the least cost. As a result, there can be agents with several goals assigned and others that will focus only on its private goals if they have.

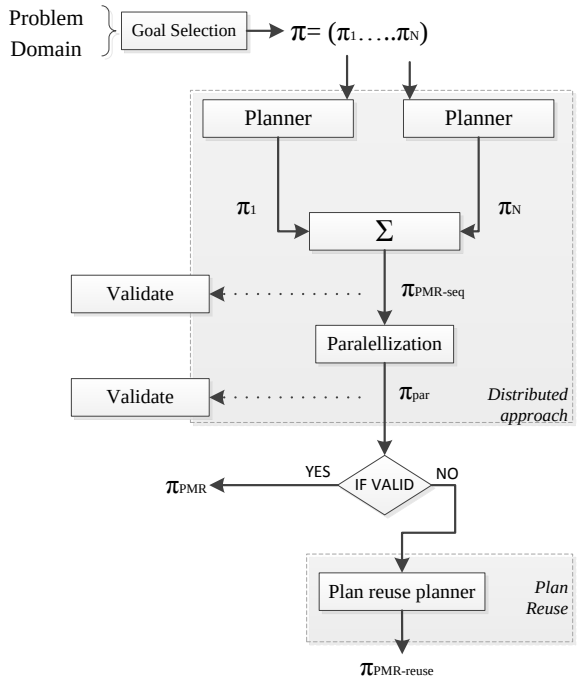


Figure 2: Architecture of PMR.

- load-balance: it performs some load balancing, not allowing an agent to be assigned more goals than the average of goals per agent. Goals are assigned first to agents as in best-cost.

Planning

In the second step of PMR, each agent invokes a planner to solve its planning task. Any state-of-the-art planner can be used for this task. Each agent will have a partial solution to the overall MAP task, composed of a total-order plan. Then, PMR concatenates all plans. This plan, a sequence of agents plans, is called $\pi_{PMR-seq}$. The concatenation process receives as an input the obfuscated description of the problem for each agent (public and private parts joined) and also the public and private goals.

After that, $\pi_{PMR-seq}$ is validated using VAL. If it is valid, most probably agents plans do not present interaction (there can still be repetitions, though) and the domain most probably will be loosely-coupled. We do not have a special focus on this paper on plan quality, so PMR returns this plan. We could easily enter an anytime search algorithm now to improve plan quality. If the plan is not valid, PMR provides it as input to the planner by reuse. More details of this in the Plan Reuse section.

Parallelization

In the third step, PMR parallelizes $\pi_{PMR-seq}$ to obtain a parallel plan π_{par}. In loosely-coupled domains, parallelization should reduce the length of the plan proportionally to the number of agents, because actions and predicates of one agent will be independent of those of the rest of agents. In tightly-coupled domains, they are not independent, and it is easier to obtain an invalid plan. Now, PMR has a plan that most probably has negative interactions. We propose to solve the task of solving plan failures by using a planner that performs planning by reuse.

Plan Reuse

In the fourth step, we can obtain the benefit of transforming π_{par} into a valid plan; the planner to be used in this step must allow PMR to start the search for a valid plan from an input plan, π_{par}. In case it can solve the planning task and the planner is sound, it will generate a new valid plan $\pi_{PMR-reuse}$. Candidate planners for this phase are LPG-ADAPT and ERRTPLAN (Borrajo and Veloso 2012).

Properties

As most work on plan merging, PMR performs suboptimal incomplete planning, since we are using suboptimal planners, working separately on subsets of goals, and choosing a subset of agents to plan for public goals. In order to make it a complete and optimal approach, we would have to use all agents, while keeping their privacy, and using optimal planners. In relation to the soundness of the algorithm, each individual (agent) plan is sound, since we are using sound planners. However, PMR cannot ensure soundness of the concatenation. But, if $\pi_{PMR-seq}$ is sound, π_{par} will be sound too. But, after invoking the plan reuse planner, PMR ensures soundness of the whole process if the plan reuse planner is sound (we use LPG-ADAPT in the experiments, that is sound).

A key issue in MAP is agents privacy. Before the parallelization phase, each agent builds its plan without sharing private information, so privacy is preserved. In the last two steps, parallelization and obtaining a valid plan from plan reuse, it would need to handle private information from agents. In our case, agents can obfuscate their private information, as it is done for MAPR. In that case, the plan reuse planner will be able to handle information related to each agent (i.e. what resource it is using, where it is placed) in order to solve the conflicts between shared resources or interests. In the current implementation, though, agents do not obfuscate that information yet, but in can be easily done given the previous obfuscation processes taking part in MAPR.

Experiments and Results

This section presents some experiments on the performance of PMR and its comparison with other planners. We show results of quality of solutions and time taken to obtain the solutions in several multi-agent domains. In our work, quality for parallel plans represents the makespan not the plan cost. In a previous paper (Borrajo 2013), we showed comparisons of a related approach, MAPR, with other state-of-the-art multi-agent planners, where those other planners did not scale well in the compared domains. Since we used very similar experimental conditions as in the other paper, it can be easily inferred that the results of the other planners will be similar.

Our configuration of PMR uses LAMA-UNIT-COST as the planner P of the algorithm. Agents use it to generate the individual plans. LAMA-UNIT-COST corresponds to the first search that LAMA performs, using greedy best first with all actions considered to have unit cost. The reason of choosing this planner is because our aim is to efficiently find plans; additionaly (as shown in the results), PMR with LAMA-UNIT-COST does not perform bad in quality. For plan reuse, we have decided to use LPG-ADAPT, an stochastic (re)planner that starts the search for a valid plan by using the input plan, π_{par}.

- **Comparing approaches.** We compare parallel plans obtained by PMR (before and after invoking LPG-ADAPT) against the ones obtained by LPG and LAMA-UNIT-COST. To evaluate the results, we used two scores of the 7th International Planning Competition (IPC): quality and time1. Quality of plans is measured as the makespan (number of parallel steps). Time is measured in seconds. We have sum the time for building $\pi_{PMR-seq}$ and π_{par} in the case of PMR and LAMA-UNIT-COST plans. Since LPG is able to build parallel plans from scratch, we use its planning time. Finally, the time for building $\pi_{PMR-reuse}$ is obtained from the time for computing π_{par} plus the time used by LPG-ADAPT to generate the new plan.

  LPG-ADAPT is called in speed mode and only cares about finding one solution. The same configuration is used for LPG. LAMA-UNIT-COST refers to the LAMA heuristic with unitary costs placed inside Fast Downward, which is the one we have used inside PMR. In order to compare PMR with LAMA, after obtaining the sequential plan of LAMA-UNIT-COST using the centralized approach of MAPR (Borrajo 2013), we perform the same parallelization step of PMR to generate the parallel plan of the LAMA-UNIT-COST plan.
- **Domains.** We have used Rovers, Zenotravel, Satellite, Transport and Port domains (Port was defined in (Borrajo 2013), while the rest are IPC domains). Each domain has 20 problems to be solved by a set of agents. The problems in the IPC, varies the number of agents. The problems of the IPC domains are the ones used in the respective IPCs. The problems of the Port domain were randomly generated for the previous paper. These domains go from loosely-coupled (like Satellite) to tightly-coupled (Port, which is similar to a multi-robot blocksworld).
- **Goal assignment.** We have used the four GA strategies: all-achievable (AA), best-cost (BC), load-balance (LB) and rest-achievable (RA).
- **Planners.** Inside PMR we have used LAMA-UNIT-COST in the distributed approach and LPG-ADAPT in the reuse phase. The planners we are comparing with are LAMA-UNIT-COST and LPG. They build parallel plans following a centralized approach.
- **Time bound.** We have used 1800 seconds to let the planners solve the problems. Most problems were solved, though Transport and Port are more difficult than the rest of domains. LPG-ADAPT solved almost all invalid problems of PMR and LAMA-UNIT-COST.

The next tables show the results of the experiments; Table 1 shows time results and the next ones quality results. Table 1 shows the summary of time scores obtained in each domain with each planner. Table 2 and 3 show the summary of quality scores obtained in each domain with each planner with sequential and parallel plans, respectively. In these tables we have used the following notation for each phase of the algorithm (Figure 2): PMR equals π_{PMR} plans; PMR-SEQ equals $\pi_{PMR-seq}$ and PMR-LPG-AD equals $\pi_{PMR-reuse}$. Again, take into account that both LPG-ADAPT and LAMA-UNIT-COST are centralized approaches that do not preserve privacy.

In relation to time results, as expected PMR scores are better than PMR-LPG-AD scores. Related to our algorithm, the best configuration is RA (rest achievable). The worse configuration is AA (all-achievable) as we expected, because in many domains, it is not valid that two or more agents achieve the same goal separately. The combined plans will not be valid most of the times. For instance, when a truck delivers a package in the Transport domain, it will change the location of the package, while the rest of trucks assume the package is in its initial state and will try to move it also.

Table 2 shows a summary of quality scores of sequential plans of PMR-SEQ and LAMA-UNIT-COST. Clearly, LAMA-UNIT-COST total quality outperforms any other configuration. This is because LAMA-UNIT-COST uses a centralized approach, while PMR, cannot build valid plans if the domain is tightly-coupled as in the Port domain.

Table 3 shows the quality scores of parallel plans. It presents a comparison of the output plans of our algorithm, PMR and PMR-LPG-AD, as well as LAMA-UNIT-COST and LPG results. If π_{par} was a valid plan (π_{PMR}), we have assigned directly its makespan to the equivalence in PMR-LPG-AD. Otherwise, LPG-ADAPT generates the new plan, and its makespan is used. We can see that the LB goal assignment strategy in PMR-LPG-AD is the best configuration. Again, the AA configuration is the worst configuration for the same reason as before. LAMA-UNIT-COST now decreases its quality, because the rest of the configurations can build valid plans. LPG is the fastest in time, but not in quality. When domains are more difficult like Transport or tightly-coupled like Port the performance of LPG decreases considerably.

This table also shows why we use LPG-ADAPT: the planner has converted with success the invalid plans of the configuration of PMR with the AA strategy into valid plans without loosing so much coverage. The AA configuration generated invalid plans in all domains except in the Satellite domain (because goals are completely independent). Also in Transport both BC and LB obtained some invalid plans. In the Port domain, as it is a tightly-coupled domain, all plans failed, except in one problem. Sometimes the quality between PMR and PMR-LPG-AD does not change, because all plans from PMR were valid and no reuse phase was needed as LB in the Satellite domain.

Related Work

Multi-Agent Planning (MAP) lies between the automated planning and multi-agent communities, with strong implications in other areas, as robotics. As was discussed in the in-

Table 1: Time score parallel plans.

Algorithms		Rovers	Zenotravel	Satellite	Transport	Port	TOTAL
PMR	AA	2.02	1.10	9.86	0.00	0.38	13.36
	BC	9.56	11.67	11.28	7.42	0.41	40.35
	LB	8.94	10.68	11.31	3.85	0.43	35.21
	RA	9.95	12.89	11.34	19.39	0.43	54.01
LAMA-UNIT-COST		19.96	10.89	9.80	8.95	5.89	55.48
PMR-LPG-AD	AA	8.84	10.01	8.63	1.51	7.74	36.72
	BC	9.56	11.67	8.83	9.84	7.91	47.80
	LB	8.94	10.68	8.80	8.45	7.94	44.81
	RA	9.95	12.89	8.85	19.39	7.84	58.92
	LPG	20.00	19.56	20.00	0.66	15.00	75.22

Table 2: Quality score of sequential plans.

Algorithms		Rovers	Zenotravel	Satellite	Transport	Port	TOTAL
PMR-SEQ	AA	3.63	2	7.66	0	1.00	14.30
	BC	19.10	18.01	16.18	9.13	1.00	63.42
	LB	18.19	14.81	14.50	4.78	1.00	53.28
	RA	17.97	17.79	17.01	18.33	1.00	72.10
LAMA-UNIT-COST		19.97	19.56	19.61	14.84	17.70	91.68

Table 3: Quality score of parallel plans.

Algorithms		Rovers	Zenotravel	Satellite	Transport	Port	TOTAL
PMR	AA	3.56	2.00	8.43	0.00	0.92	14.90
	BC	15.49	14.42	11.03	7.41	0.92	49.27
	LB	19.46	17.93	18.97	4.96	0.92	62.23
	RA	19.46	12.19	10.32	8.89	0.92	51.78
LAMA-UNIT-COST		16.78	18.56	13.35	14.31	10.17	73.17
PMR-LPG-AD	AA	11.01	8.69	8.43	0.89	7.57	36.58
	BC	15.49	14.42	11.03	11.05	7.27	59.27
	LB	19.46	17.93	18.97	11.15	7.24	74.74
	RA	19.46	12.19	10.32	8.89	6.82	57.69
	LPG	15.55	16.39	15.33	0.44	13.74	61.45

troduction, approaches range from centralized to distributed planning. In case of distributed planning, some papers dealt with a distributed coordinated approach when generating plans (Nissim and Brafman 2013; Jonsson and Rovatsos 2011; Torreño, Onaindia, and Sapena 2014), while others preferred to delay coordination and perform plan merging after generating the individual plans (Foulser, Li, and Yang 1992). As an example of plan merging, in (Foulser, Li, and Yang 1992) its authors developed some algorithms to merge a plan decomposed previously in subplans in order to solve a task. They present both efficient and optimal heuristic algorithms, while we are interested in satisficing algorithms. Also, Mali in (Mali 2000) proposed plan merging and plan reuse as a new way to obtain satisficing plans. Mali identifies two types of merging: contiguous and partially order; PMR only focuses on the second one. The main difference with his work is that we combine plan merging and plan reuse to generate valid plans. In Mali's work if the merging needed an extra action that was not present in the individual plans (due to some strong interaction among plans), it was not able to generate it. Another difference is that PMR can handle plans where the same action appears in several individual plans, while Mali's approach could not.

Later, Britanik in (Britanik and Marefat 1995) proposed to do plan merging in HTN planning. Merging appears in different levels of abstraction by decomposing a plan in subplans. If a replanning phase is needed, it will be easier to apply because of the independency of the plans. None of these works have taken into account the agents privacy. Our approach focuses on classical planning, but our aim is to use plan merging to obtain fast a parallel plan that is not necessarily valid in its first instance. Also, our goal is to maintain the agents privacy. We do not have a replanning phase, but a reuse phase where the parallel plan is received as an input.

Finally, in (Brafman and Domshlak 2006) Brafman and Domshlak propose a decomposition method of the planning domain. Instead, PMR uses the agents to decompose the problem.

Conclusions

We have presented PMR, an algorithm capable of solving a multi-agent planning task using plan-merging and plan-reuse techniques. First, it uses a distributed approach to let agents build each plan individually, and then it concatenates each plan to obtain a parallel plan. We also showed that PMR does not ensure soundness, so if the parallel plan was not valid, LPG-ADAPT generates a new valid plan based on the invalid parallel plan.

After comparing the plans obtained with PMR and the ones from LPG and LAMA-UNIT-COST, the reuse phase improves coverage over previous steps. PMR's BC and LB configurations obtain the best results, comparable to those of LPG and LAMA-UNIT-COST. In fact, LB improves the quality scores of the two planners.

As future work, we plan to compare our algorithm with other factored planning approaches, to understand the advantage of decomposing based on agents over other alternative approaches for decomposing a planning task.

Acknowledgments

This work has been partially supported by Spanish MICINN projects TIN2011-27652-C03-02.

References

Borrajo, D., and Veloso, M. M. 2012. Probabilistically reusing plans in deterministic planning. In *In Proceedings of ICAPS'12 workshop on Heuristics and Search for Domain Independent Planning*. AAAI Press.

Borrajo, D. 2013. Plan sharing for multi-agent planning. In Nissim, R.; Kovacs, D. L.; and Brafman, R., eds., *Preprints of the ICAPS'13 DMAP Workshop on Distributed and Multi-Agent Planning*, 57–65.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *In Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-2006*, 809–814.

Brafman, R. I., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

Britanik, J., and Marefat, M. 1995. Hierarchical plan merging with application to process planning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, 1677–1684. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Foulser, D.; Li, M.; and Yang, Q. 1992. Theory and algorithms for plan merging. *Artificial Intelligence* 57(2-3):143–181.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS'06)*, 212–221.

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

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In Khoshgoftaar, T. M., ed., *ICTAI 2004: 16th IEEE International Conference on Tools with Artificial Intelligence*, 294–301.

Jonsson, A., and Rovatsos, M. 2011. Scaling up multiagent planning: A best-response approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 114–121.

Mali, A. D. 2000. Plan merging & plan reuse as satisfiability. In *Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning. ECP'99*, 84–96. Springer-Verlag.

Nissim, R., and Brafman, R. I. 2013. Cost-optimal planning by self-interested agents. In *Proceedings of AAAI'13*.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014. A Flexible Coupling Approach to Multi-Agent Planning under In-

complete Information. *Knowledge and Information Systems* 38(1):141–178.

Veloso, M. M.; Pérez, M. A.; and Carbonell, J. G. 1990. Nonlinear planning with parallel resource allocation. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 207–212. San Diego, CA: Morgan Kaufmann.

Improving Uncoordinated Collaboration in Partially Observable Domains with Imperfect Simultaneous Action Communication

Aris Valtazanos
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB, UK
a.valtazanos@ed.ac.uk

Mark Steedman
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB, UK
steedman@inf.ed.ac.uk

Abstract

Decentralised planning in partially observable multi-agent domains is limited by the interacting agents' incomplete knowledge of their peers, which impacts their ability to work jointly towards a common goal. In this context, communication is often used as a means of observation exchange, which helps each agent in reducing uncertainty and acquiring a more centralised view of the world. However, despite these merits, planning with communicated observations is highly sensitive to communication channel noise and synchronisation issues, e.g. message losses, delays, and corruptions. In this paper, we propose an alternative approach to partially observable uncoordinated collaboration, where agents simultaneously execute and communicate their *actions* to their teammates. Our method extends a state-of-the-art Monte-Carlo planner for use in multi-agent systems, where communicated actions are incorporated directly in the sampling and learning process. We evaluate our approach in a benchmark multi-agent domain, and a more complex multi-robot problem with a larger action space. The experimental results demonstrate that our approach can lead to robust collaboration under challenging communication constraints and high noise levels, even in the presence of teammates who do not use any communication.

Introduction

Collaborative planning is an important challenge for many interactive systems, where multiple agents must work together to achieve a common goal. This problem becomes harder when agents do not have the benefit of centralised coordination, or when the task involves collaboration with a priori unknown teammates. For example, consider a rescue scenario where various robots programmed by different engineers are deployed to a disaster site in an emergency situation. In this setting, generating a commonly agreed plan of actions may be infeasible due to tight time constraints and limited knowledge of the environment. Instead, the robots may be forced to plan from an egocentric perspective, by using their own internal models to select robust actions.

When the above constraints on collaboration arise, agents must reason about the actions of their peers by gathering and processing data on their behaviour. In a general multi-agent planning setting with no centralised coordination, there are two types of input that can be acquired by an agent:

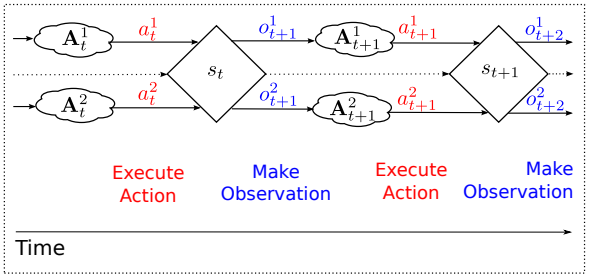
1. Direct **observations** of the teammates, e.g. images from a camera, sonar readings, or other sensory inputs.
2. Inter-agent **communication**, i.e. messages received from teammates about their own (past or future) actions, observations, plans, or intentions.

Each of these input types is significant for uncoordinated collaboration, but also carries its own challenges and limitations. On the one hand, sensory observations are collected and processed internally by each agent, so they are not generated by unknown external protocols or mechanisms. However, many domains of practical interest are characterised by partial and/or limited observability, so agents may not be able to view their teammates (reliably and noiselessly) at all times. On the other hand, a communicated message provides direct insight on the planning process used by the sending agent, thus helping the receiving agent in selecting its own actions with regard to the overall team goal. However, limited bandwidth or poor synchronisation issues may lead to dropped or delayed messages during an interaction. Furthermore, communication channels may be noisy or unreliable, giving rise to misinterpreted (or uninterpreted) messages that also impact an agent's knowledge of its peers.

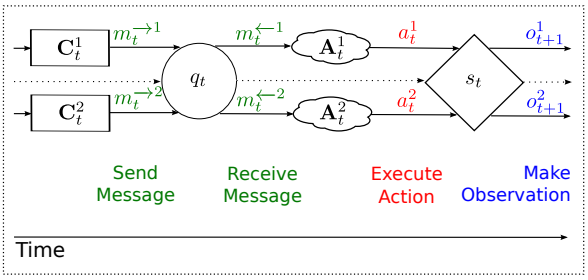
In light of the above constraints, an important challenge in uncoordinated collaboration under partial observability lies in combining the relative merits of observation- and communication-based reasoning. Planning under limited observations has been widely studied using the Partially Observable Markov Decision Process (POMDP) formulation (Kaelbling, Littman, and Cassandra 1998), which however does not explicitly model communication (Figure 1(a)). By contrast, communication-based planning in multi-agent systems is a more open-ended problem, which is typically concerned with the following issues:

1. **When** and **how** to communicate.
2. **What** to communicate.

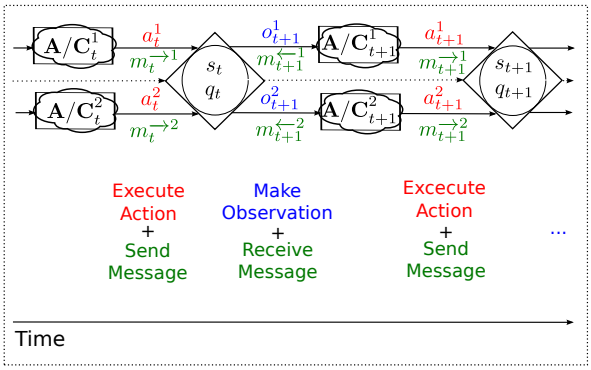
With regard to the first issue, there is a distinction between implementations assuming *perfect synchronisation* between communication and planning phases (where agents can reliably exchange messages before selecting their actions, as in Figure 1(b)), and those accounting for *stochastic communication* (where messages can be lost or delayed). With regard to *what* to communicate, a commonly employed approach is *observation-based* communication (as in the work of Pynadath and Tambe (2002)), where agents exchange their *most recent* observations. The motivation behind this choice is



(a) Planning with no communication (as in standard decentralised POMDP planning implementations).



(b) Planning with distinct (synchronised) action selection and communication phases.



(c) Planning with simultaneous action selection and communication phases (the approach followed in this paper).

Figure 1: Sketch drawings of different approaches to combining decentralised planning and communication in partially observable multi-agent domains. Illustrations are given for the two-agent case, with superscripts ^{1} and ^{2} being the agent indices, and subscripts denoting time. *Clouds:* Action selection/planning steps ($\mathbf{A}$). *Rectangles:* Communication/message selection steps ($\mathbf{C}$). *Diamonds:* State updates. *Circles:* Communication updates. Other notation: actions – a, observations – o, world states – s, message queues – q, sent messages – $m^{\rightarrow}$, received messages – $m_t^{\leftarrow}$.

that agents obtain an approximate world model by combining the locally transmitted views, thus effectively reducing collaboration to a centralised planning problem.

Despite these advantages, we also note some important limitations of observation-based communication. First, planning becomes sensitive to stochastic communication, as delayed or dropped observation messages inevitably lead to an incomplete or outdated world model. Second, even when communication is perfectly synchronised, there is an underlying assumption that all agents use the same planning mechanism (and thus interpret each other's observations identically), which however breaks down when heterogeneous teammates are called to collaborate (as in the rescue scenario example introduced earlier). Third, reasoning about other agents' observations effectively means modeling their own *beliefs* and uncertainty about the world state, which increases the depth of reasoning and thus also the complexity of the planning process.

In this paper, we propose a novel *action-based communication* model for uncoordinated collaboration in partially observable domains. Our approach extends a state-of-the-art online POMDP Monte-Carlo planner with a simple communication protocol, where agents execute and broadcast their selected actions *simultaneously* (Figure 1(c)). Agents maintain a distribution (defined in terms of their *own* beliefs) over selected teammate actions, which is updated when a new message is received. The planner then uses this distribution as a prior in action sampling during Monte-Carlo iterations, and to perform a new type of *factored* policy learning, which decouples observation- and message-based value updates.

As illustrated in Figure 1(c), our protocol implies that transmitted messages are only received *after* the current planning cycle. Thus, even when the communication channel is perfect and noiseless, agents will *always* have delayed information on their peers. This motivates a looser coupling between communication and planning, which, as we demonstrate in our results, makes our approach more robust to three types of noise:

1. Message **losses**.
2. Message **delays**.
3. Message **corruptions/misinterpretations**.

The latter type of noise has received less attention in partially observable multi-agent planning, but we argue that it is particularly important when considering collaboration with heterogeneous agents, such as humans or human-controlled robots. These settings typically involve complex speech generation and recognition processes that significantly constrain communication within a team.

Another distinguishing feature of our approach is that agents do not exchange their observations, and thus also do not explicitly model each other's beliefs and planning mechanisms. This keeps the computational complexity of our approach low and scalable to challenging domains.

In the remainder of this paper, we first review related ideas and techniques from the literature, and we subsequently present our methodology, describing our planning algorithms and communication protocol. We then evaluate our approach in two multi-agent domains; a benchmark box-pushing problem with a small action space, and a more challenging multi-robot kitchen planning scenario. Our results demonstrate that planning with action communication outperforms non-communicative implementations under most noise configurations, while requiring comparable computation time. We conclude by summarising our key contributions and suggesting possible future directions.

Related Work

Single-agent planning

Planning in partially observable single-agent domains is usually described in terms of a Partially Observable Markov Decision Process (POMDP) (Kaelbling, Littman, and Cassandra 1998), $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{R} \rangle$, where $\mathcal{S}, \mathcal{A}, \mathcal{O}$ are the state, action, and observation sets, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition function, $\mathcal{Z} : \mathcal{S} \times \mathcal{O} \times \mathcal{A} \mapsto [0, 1]$ is the observation function, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \Re$ is the reward function giving the expected payoff for executing an action.

POMDPs can be used to model a wide range of decision problems. However, analytical solutions are known to be hard to compute (Papadimitriou and Tsitsiklis 1987), with several problems requiring hours or even days to solve exactly. This is a restricting factor for systems with tight computational constraints and varying task specifications.

The complexity of finding offline analytical solutions has led to the development of *online* POMDP planning methods, which only consider the current state of the interaction and use limited computation time. Partially Observable Monte Carlo Planning (POMCP) (Silver and Veness 2010) employs Monte-Carlo Tree Search to *sample* the problem space efficiently. This method models action selection as a *multi-armed bandit* problem, by initially estimating the value of random action sequences (referred to as *rollouts*), and then balancing exploration and exploitation through the Upper Confidence Bound (UCB) heuristic (Kocsis and Szepesvári 2006). POMCP has been successfully applied to problems with large branching factors (Gelly et al. 2012), and implemented in a winning entry of the 2011 International Planning Competition (Coles et al. 2012).

Multi-agent planning without communication

A Decentralised POMDP (Dec-POMDP) (Bernstein et al. 2002) is a generalisation of a POMDP to multi-agent systems, defined as $\langle \mathcal{I}, \mathcal{S}, \vec{\mathcal{A}}, \vec{\mathcal{O}}, \mathcal{T}, \mathcal{Z}, \mathcal{R} \rangle$, where $\mathcal{I} = \{1...n\}$ is the set of agents, $\vec{\mathcal{A}} = \times_{i \in \mathcal{I}} \mathcal{A}^i$ is the set of joint actions $\vec{a} = \langle a_1, ..., a_n \rangle$, defined as the Cartesian product of the agents' individual action sets $\mathcal{A}^i$, $\vec{\mathcal{O}} = \times_{i \in \mathcal{I}} \mathcal{O}^i$ is similarly the set of joint observations, with $\mathcal{T}$, $\mathcal{Z}$, and $\mathcal{R}$ defined as in POMDPs, with $\vec{\mathcal{A}}$ and $\vec{\mathcal{O}}$ substituting $\mathcal{A}$ and $\mathcal{O}$.

Compared to POMDPs, Dec-POMDPs carry the additional limitation that action and observation spaces grow exponentially with the number of agents, thus also being intractable. Furthermore, fast single-agent methods like POMCP cannot be directly extended to Dec-POMDPs, due to the added constraint of reasoning about joint observations and beliefs. In this paper, we describe an alternative, egocentric method of adapting POMCP to multi-agent system constraints. Each agent keeps track of and updates values over only its own history and observation space, with teammate actions modeled at the rollout sampling level. This keeps the complexity of the planning process low and scalable to larger and more complex planning spaces.

Multi-agent planning with communication

In their general form, the POMDP and Dec-POMDP formulations do not explicitly model communication between agents. To address this issue, several extensions combining decentralised planning and message passing have been proposed. One of the earlier such approaches is the Communicative Multi-agent Team Decision Problem (Pynadath and Tambe 2002), which presents a general framework for teamwork with instantaneous communication. However, this model assumes distinct pre-communication and post-communication phases (similarly to Figure 1(b)) and perfect noiseless channels without delays and losses.

Becker, Lesser, and Zilberstein (2005) consider communication with associated costs in a Decentralised MDP framework, where agents must additionally decide when to transmit their local *states* to their peers. This concept is extended to partially observable domains by Roth, Simmons, and Veloso (2005), leading to reasoning over joint beliefs based on intermittently transmitted local observations. Despite factoring communication costs, both of these works also assume reliable communication channels, through which agents are able to merge their local observations (or states) and construct a more complete approximation of the world.

Planning with communication costs has also been studied in the context of *coordinated* multi-agent reinforcement learning (Zhang and Lesser 2013). This method uses a *loss rate threshold* to select sub-groups of agents that will coordinate their actions (and communicate) at each time step. Despite addressing concerns of systems with larger numbers of agents, this work makes stronger assumptions on inter-agent coordination, while also not considering noise (and actual message losses) in the communication channel.

The problem of decentralised planning with *imperfect* communication has recently received more attention in the literature. Within the Dec-POMDP framework, Bayesian game techniques (Oliehoek, Spaan, and Vlassis 2007) and tree-based solutions (Oliehoek and Spaan 2012) have been proposed to deal with *one-step message delays*. This is extended to account for stochastic delays that can be longer than one time step (Spaan, Oliehoek, and Vlassis 2008). Our simultaneous communication model (Figure 1(c)) aims to address similar effects, but does not assume any explicit bounds on message delays. Furthermore, we also consider other types of communication noise such as message losses (which are effectively analogous to infinite-time delays).

Wu, Zilberstein, and Chen (2011b) introduce a model of bounded message-passing, where the communication channel may be periodically unavailable. In this context, two distinct protocols are evaluated; the first postpones communication until the channel becomes available again, and the second drops the communication attempt entirely. While these constraints are similar to the ones we consider, we also note some important differences. First, the bounded communication model uses separate communication and action phases, whereas we adopt a more constrained simultaneous approach (Figure 1). Second, the above protocols assume that agents know *when* the communication channel is unavailable; by contrast, our method makes no assumptions on when, if, or how transmitted messages will reach other teammates.

A common feature of all the above works is that agents communicate their local *observations* to each other, with

the goal of combining them and forming a more complete world model. As discussed in the introduction of this paper, we adopt a different, *action*-based communication protocol, which does *not* aim to "centralise" a decentralised decision problem through observation exchange and joint-belief reasoning. Instead, agents maintain their own incomplete views of the world, and use (any) communicated actions received from their teammates to bias their own, egocentric planning process. As we demonstrate in our results, this protocol maintains a robust performance even under high levels of communication channel noise. Moreover, our approach is also tolerant to novel types of noise, such as message corruption, which have so far received little attention in decentralised planning under partial observability.

Collaboration without prior coordination

Another common underlying aspect of the works presented in the previous section is collaboration between *identical* agents. However, many of these approaches break down when the domains feature heterogeneous agents with different planning, reward, or world modeling processes. To address this issue, our method draws inspiration from the *ad-hoc* teamwork problem (Stone et al. 2010), which considers collaboration without pre-coordination in the presence of unknown teammates. In this context, the POMCP algorithm has been combined with Markov games (Wu, Zilberstein, and Chen 2011a) and transfer learning (Barrett et al. 2013b) to generate team-level strategies. However, both of these works assume full world observability and do not involve inter-agent communication.

A communication protocol for ad-hoc teamwork has been proposed by Barrett et al. (2013a), where message selection is integrated within the planning process. In particular, each agent has a fixed set of communicative messages that are synthesised through the POMCP multi-armed bandit framework. Despite taking some important first steps towards combining planning and communication with heterogeneous agents, this work assumes full world observability and noiseless channels, while also using distinct communication and action phases. To our knowledge, our method is the first to address the combined existence of several of the challenges described so far, i.e. uncoordinated collaboration with unknown teammates in partially observable domains, in the presence of imperfect communication.

Method

In this section, we first provide an overview of POMDP and Monte-Carlo planning, summarising some key concepts from Silver and Veness (2010). Then, we extend the single-agent POMCP definitions to model egocentric planning in multi-agent systems. We subsequently present our communication protocol, and then describe our approach to planning with communicated actions. We conclude by providing detailed algorithms for our implementation.

Planning in single-agent POMDPs

Preliminaries An agent acting in a partially observable domain cannot directly observe the state of the world, s_t, but only knows a history of past actions and observations up to the current time t, $h_t = \{o_0, a_0, \ldots, o_t, a_t, o_{t+1}\}$, and plans with respect to the belief $B(s, h)$, which is a history-dependent distribution over states. A policy $\pi(h, a)$ is a mapping from histories to actions, and the *return* $R_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_k$ is the obtained reward starting at time t, where $0 < \gamma \leqslant 1$ is a discount factor, and each r_k is drawn from the reward function $\mathcal{R}$. The *value function* $V^\pi(h) = \mathbb{E}[R_t|h]$ is the expected return under π starting at history h, and $V^*(h) = \max_\pi V^\pi(h)$ is the optimal value function. Additionally, $Q^\pi(h, a)$ is the value of taking action a after history h, and then following policy π.

Monte-Carlo planning Due to the complexity associated with computing V^* exactly, POMCP approximates this value through sampling-based forward search from the current history h. The planner uses a black-box simulator $(s_{t+1}, o_{t+1}, r_{t+1}) \sim \mathcal{G}(s_t, a_t)$ that generates successor values given the current state and action. The value of a state s is approximated by the mean return of n *simulations*, or *plan samples*, $V(s) = \frac{1}{n} \sum_{i=1}^{n} R_i$, each searching the problem space over a fixed time horizon $\mathcal{H}$. Starting with 0 values, the planner also maintains visitation counts $N(h), N(h, a)$ and Q-value estimates $Q(h, a)$ for each history-action pair, which are updated during plan sampling; visitation counts are incremented by 1 each time a history or history-action pair is sampled, and Q-values are updated as $Q(h, a) \leftarrow Q(h, a) + \frac{R - Q(h,a)}{N(h,a)}$, where R is the return of the most recent plan sample. When a history h has not been visited before, actions are chosen randomly based on a *rollout* policy, $a \sim \pi_{rollout}(h)$. Otherwise, the optimal action is selected as

$$a^* = \arg\max_{a \in \mathcal{A}} Q(h, a) + c\sqrt{\log(N(h))/N(h, a)}, \quad (1)$$

using the UCB heuristic with an exploration constant c.

Egocentric POMCP for multi-agent systems

Extending POMCP heuristics to multi-agent systems is not straightforward due to the existence of joint actions and observations. For fully observable systems, Eq. 1 can be rewritten as $\vec{a}^* = \arg\max_{\vec{a} \in \vec{\mathcal{A}}} Q(s, \vec{a}) + c\sqrt{\log(N(s))/N(s, \vec{a})}$ (Wu, Zilberstein, and Chen 2011a), replacing histories with states and single-agent actions with joint ones. Unfortunately, this does not apply to partially observable domains with no communication because agents cannot observe joint histories $\vec{h}$ (and actions $\vec{a}$).

To overcome this problem (and avoid maintaining expensive beliefs over the beliefs of others), we restrict our sample updates to single-agent N and Q values as in the original POMCP framework. However, we modify the rollout policy to generate random joint actions, $\vec{a} \sim \pi_{rollout}(h)$, though it is still parametrised only by the planning agent's history h. Similarly, we parametrise the black-box simulator in terms of joint actions, $(s_{t+1}, o_{t+1}, r_{t+1}) \sim \mathcal{G}(s_t, \vec{a}_t)$, though it still generates observations and rewards for the planning agent only. These modifications can be implemented at minimal additional computational cost, while also not making any assumptions about other agents' beliefs and histories.

Simultaneous Action Communication

Communication protocol and structures Despite providing support for decision-making in the presence of other agents, the above definitions do not model communication within a team. To address this issue, we define a simple protocol through which agents can communicate with each other. In particular, let $\mathcal{M}$ denote the set of messages that can be exchanged between agents, $m^{\rightarrow}$ a message *sent* by an agent to its teammates, and $m^{\leftarrow}$ a *received* message. We assume a simple broadcasting protocol, where the world state, s, is augmented with a *message queue*, q, containing all the currently available messages. When an agent sends a message $m^{\rightarrow}$, it simply adds it to the front of q; when an agent receives a message $m^{\leftarrow}$, it marks it for removal and $m^{\leftarrow}$ is erased from q at the end of the current time step.

Message selection, exchange, and interpretation As discussed in previous sections, one distinguishing feature of our approach is that agents communicate their actions (and not their observations), and they do so *simultaneously* with action execution (as illustrated in Figure 1(c)). In this context, an agent selects the action a_t to be executed at time t, an deterministically sets its upcoming message to $m_t^{\rightarrow} = a_t$. Thus, the message set is identical to the action set, i.e. $\mathcal{M} = \mathcal{A}$. Furthermore, we can straightforwardly extend the action rollout policy, $\vec{a} \sim \pi_{rollout}(h)$, to obtain the *joint message rollout* policy, $\vec{m}^{\rightarrow} \sim \mu_{rollout}(\vec{a}) = \vec{a}$.

Similarly to actions, agents receive messages from their teammates simultaneously to making observations on the state of the world. At every observation/message reception phase, each agent receives a set, $\{m^{\leftarrow}\}$, of up to $n-1$ messages, where n is the size of the team (so at most one message per teammate is received). However, the size of $\{m^{\leftarrow}\}$ may be potentially smaller when messages are delayed or dropped. The received messages are interpreted under the assumption that all agents are communicating their actions; we use a simple procedure $a \leftarrow \texttt{ParseAction}(m)$ that converts a message m to an action a. When the channel is reliable, `ParseAction` will return the (correct) action that was originally transmitted by the sending agent. Nevertheless, as discussed in the following section, we also consider the case where messages are corrupted during transmission and thus interpreted incorrectly at the receiving end.

Putting everything together, the black-box $\mathcal{G}$ with simultaneous communication and state updates is rewritten as

$$(s_{t+1}, q_{t+1}, o_{t+1}, r_{t+1}, \{m_{t+1}^{\leftarrow}\}) \sim \mathcal{G}(s_t, q_t, \vec{a}_t, \vec{m}_t^{\rightarrow}) \quad (2)$$

Modeling imperfect communication Our protocol can be extended to account for different types of imperfect communication. When modeling message losses, a transmitted message $m_t^{\rightarrow}$ is dropped with probability $0 \leqslant p(loss) \leqslant 1$, in which case the queue q remains unchanged. For message delays, $m_t^{\rightarrow}$ is added with probability $p(delay)$ to q *after* the other updates for step t are completed, which means that it cannot be used by its teammates at decision step $t+1$. Thus, our notion of delay is ***different*** to definitions assuming distinct action and communication phases. In our framework, ***all*** messages by default arrive with a one (planning) step delay, so our definition of delay refers to an ***additional*** communication lag (leading to an overall delay of at least two planning steps). Finally, a transmitted message is corrupted with probability $p(corrupt)$, in which case the receiving agent interprets it as an action other than the one that was originally sent. For the latter type of noise, the number of possible misinterpretations grows with the action set size.

Planning with Communicated Actions

One important open question in our framework is how to use communicated messages to improve the quality of selected actions. To address this issue, we propose two extensions to the original egocentric POMCP framework. First, we introduce a distribution over communicated messages, and use it as a bias in the teammate action sampling process. Second, we define and learn Q-values over the *message* space, thus obtaining a factored approach to action selection.

Teammate action sampling In the non-communicate egocentric POMCP variant, teammate actions are always sampled based on the random rollout policy $\pi_{rollout}$. However, when communication is available, the received messages can provide better insight on the actions chosen by the other agents. To exploit this feature, we introduce a distribution $A'(h, a)$ over communicated teammate actions for every (single-agent) history h and action a. We model $A'(h, a)$ as an unweighted particle filter that is progressively populated from the received messages (similarly to how the belief distribution $B(h)$ is updated from the generated state samples). When $A'(h, a)$ is non-empty, teammate actions are sampled directly from this distribution, otherwise we use $\pi_{rollout}$ as in the non-communicative approach. Thus, action selection is biased towards the information extracted from the received teammate messages, and the rollout policy serves as a fall-back when communication is limited.

Factored value learning and action selection We model communicated messages a special type of observation, over which a separate set of Q-values is learned and used in action selection. In particular, we define a value $Q(h, a, m)$ (and an associated visitation count $N(h, a, m)$) for every history h, action a, and message m, thus introducing an additional layer in the policy learning hierarchy. Like regular Q-values, message values are updated based on the return R generated by each plan sample, i.e. $Q(h, a, m) \leftarrow Q(h, a, m) + \frac{R - Q(h,a,m)}{N(h,a,m)}$. Moreover, Eq. 1 is updated as

$$a^* = \arg\max_{a \in \mathcal{A}} \big[Q(h, a) + \max_{m \in \mathcal{M}} Q(h, a, m) + c\sqrt{\log(N(h))/N(h, a)} \big] \quad (3)$$

to incorporate the learned message values. This leads to a factored learning and action selection procedure, where the planning agent performs distinct learning updates for the different types of input acquired during the interaction.

Summary of Algorithms

We conclude this section by providing implementations for all the procedures described so far. Algorithm 1 summarises the high-level search algorithm; when a history h has not been visited before, states are sampled from the initial state

Algorithm 1: `SearchWithCommunication`(h)

for $i \leftarrow 1$ **to** *NumPlanSamples* **do**
  if $h = empty$ **then** $s \sim \mathcal{I}_S,\ q \leftarrow \varnothing$
  else $\langle s, q \rangle \sim B(h)$
  `Simulate`$(s, q, h, 0)$
return $\arg\max_{a \in A} \left[Q(h,a) + \max_{m \in \mathcal{M}} Q(h,a,m) \right]$

Algorithm 2: `SelectAction`(h)

return $a^* \leftarrow \arg\max_{a \in \mathcal{A}} \left[Q(h,a) + \max_{m \in \mathcal{M}} Q(h,a,m) + c \cdot \sqrt{\frac{\log(N(h))}{N(h,a)}} \right]$

Algorithm 3: `Rollout`(s, q, h, d)

if $d \geqslant \mathcal{H}$ **then return** 0
$\vec{a} \leftarrow \langle a, a' \rangle \sim \pi_{rollout}(h),\ \vec{m}^{\rightarrow} \sim \mu_{rollout}(\vec{a})$
$\langle \acute{s}, \acute{q}, o, r, \cdot \rangle \sim \mathcal{G}(s, q, \vec{a}, \vec{m}^{\rightarrow})$
return $r + \gamma \cdot$ `Rollout`$(\acute{s}, \acute{q}, hao, d+1)$

Algorithm 4: `Simulate`(s, q, h, d)

if $d \geqslant \mathcal{H}$ **then return** 0
if $N(h) = 0$ **then**
  forall $a \in \mathcal{A}$ **do**
    $N(h,a) \leftarrow 0,\ Q(h,a) \leftarrow 0,\ A'(h,a) \leftarrow \varnothing$
    forall $m \in \mathcal{M}$ **do**
      $N(h,a,m) \leftarrow 0,\ Q(h,a,m) \leftarrow 0$
  return `Rollout`(s, q, h, d)
$a \leftarrow$ `SelectAction`(h)
if $A'(h,a) \neq \varnothing$ **then** $a' \sim A'(h,a)$
else $\langle \cdot, a' \rangle \sim \pi_{rollout}(h)$
$\vec{m}^{\rightarrow} \sim \mu_{rollout}(\langle a, a' \rangle)$
$\langle \acute{s}, \acute{q}, o, r, \{m^{\leftarrow}\} \rangle \sim \mathcal{G}(s, q, \langle a, a' \rangle, \vec{m}^{\rightarrow})$
$R \leftarrow r + \gamma \cdot$ `Simulate`$(\acute{s}, \acute{q}, hao, d+1)$
$B(h) \leftarrow B(h) \cup \langle s, q \rangle,\ N(h) \leftarrow N(h) + 1$
$N(h,a) \leftarrow N(h,a)+1, Q(h,a) \leftarrow Q(h,a) + \frac{R - Q(h,a)}{N(h,a)}$
forall $m \in \{m^{\leftarrow}\}$ **do**
  $N(h,a,m) \leftarrow N(h,a,m) + 1$
  $A'(h,a) \leftarrow A'(h,a) \cup$ `ParseAction`(m)
  $Q(h,a,m) \leftarrow Q(h,a,m) + \frac{R - Q(h,a,m)}{N(h,a,m)}$
return R

distribution $\mathcal{I}_S$, and the message queue is empty. Algorithm 2 recaps the communication-based action selection formulation, and Algorithm 3 gives the random rollout sampling procedure. Finally, Algorithm 4 illustrates the main simulation algorithm, where Q-values are initialised and updated.

Results

We evaluate our approach in two multi-agent domains; a benchmark cooperative box-pushing problem from the Dec-POMDP literature, and a more complex multi-robot kitchen scenario with a significantly larger action space. The domains are noisy, so actions and observations are perturbed with 0.1 probability. In both problems, we compare a decentralised POMCP agent with simultaneous action communication (denoted **SAC**) to an identical agent with no communication (**NoComm**); both agents follow the planning approach defined in the previous section, but **NoComm** does not learn or use any message Q-values.

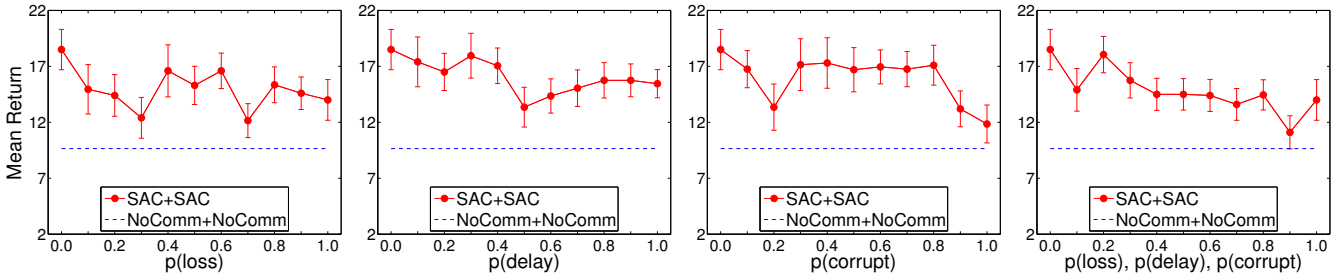
We assess the two algorithms in two-agent teams, dividing our experiments for each domain in two phases. In the first, *same-agent team* phase, we pair **CAC** and **NoComm** with an identical agent, and compare the resulting teams under varying message loss, delay, and corruption probabilities. We test for the cases where each threshold ($p(loss)$, $p(delay)$, $p(corrupt)$) is modified independently, and the case where all types of noise are combined. In the second, *heterogeneous-agent team* phase, we fix the three noise thresholds to 0.1 (so they are equal to the action and observation noise probabilities), and compare performance in collaboration with other, unknown agents. In this context, the candidate teammates are an agent selecting random actions (**Rand**), and a *problem-specific* human-designed agent (**HumDes**) running a robust hand-coded algorithm. Both **Rand** and **HumDes** can communicate their actions to their teammates, though their behaviour does not make any assumptions about the availability of communication.

To demonstrate the generality of our approach, we use the same experiment parameters in both problems. For each team, we average results over 100 runs with 1024 plan samples per decision step, recording the **mean return** (the reward achieved by the team after each run) and the average computation **time** *per team per step*. We set the time horizon to $\mathcal{H} = 20$ and the exploration constant to $c = r_{max}$, where r_{max} is the maximum reward of the domain. Experiments were run on a dual core 3GHz PC with a 4GB RAM.

Cooperative box-pushing

In the cooperative box-pushing domain (Seuken and Zilberstein 2007), agents interact in a walled grid with one large and two small boxes. Agents get a reward of +10 for pushing a small box to the edge of the grid and +100 for doing this for the large box. However, the large box can be moved only if simultaneously pushed by both agents. Each agent has 4 actions (*move*, *turn_left*, *turn_right*, *stay*) and can only see the square to its front, with the possible observations being *empty*, *other_agent*, *wall*, *small_box*, *large_box*. Each agent gets a reward of -0.1 for every step taken, and -5 for bumping into a wall, its teammate, or a box it cannot move. When any box reaches the edge, the problem resets to the start state and the interaction repeats until the time horizon is reached.

The box-pushing results for same-agent teams under different types of communication noise are presented in Figure 2. The **SAC + SAC** team is seen to outperform the **NoComm** variant under all possible noise thresholds, even when the communication channel is always unavailable or unreliable. Moreover, the performance is similar across the different types of noise (and the case where all types of noise combine), thus indicating that our method is not sen-



(a) Message losses (b) Message delays (c) Message corruptions (d) All types of noise combined

Figure 2: Box-pushing domain - comparison of same-agent teams (**SAC + SAC** and **NoComm + NoComm**) for different types of communication channel noise. $p(loss)$: probability of message loss. $p(delay)$: probability of message delay. $p(corrupt)$: probability of message corruption. (a)-(c): Returns obtained under a single type of noise - the other probabilities are set to 0. (d): All types of noise combined (with $p(loss) = p(delay) = p(corrupt)$ in each case).

sitive to any specific irregularities. Thus, the simultaneous action communication approach benefits from the exchange of messages when the channel is reliable, while not being impacted by message losses, delays, or corruptions even under severely restricted communication conditions.

An experimental evaluation of decentralised planning with communication in the box pushing domain has also been conducted by Wu, Zilberstein, and Chen (2011b). In their results, they report considerably higher positive returns for most noise thresholds, which however drop to negative values when the channel is always unavailable (whereas our method still manages to achieve a positive mean return). Nevertheless, a direct comparison with simultaneous action communication is problematic for two reasons. First, as discussed in the related work section, their approach uses distinct action execution and communication phases, where successfully transmitted messages always provide up-to-date information on teammate observations. In our framework, even where there is no additional noise, all messages arrive with a one-step delay. Thus, our experimental setting introduces considerably harder constraints on collaboration that are not fully captured by the distinct phase model[1]. Second, their method first solves a centralised MDP version of the problem, and then uses it as a heuristic in the decentralised algorithms, thus improving planning performance. By contrast, our approach uses no such heuristics, following a fully uncoordinated planning approach that does not incorporate any prior centralised knowledge.

The results for collaboration with heterogeneous agents in the box-pushing domain are given in Figure 3. Compared to **NoComm**, the **SAC** agent achieves better returns when paired with all other teammates. Moreover, the **SAC + NoComm** team outperforms both the **SAC + SAC** and the **NoComm + NoComm** combinations, thus indicating that our

[1]In practice, the distinct and simultaneous phase models are aligned only in the maximal 1.0 loss rate case, when all messages are dropped in both frameworks (in this case, only our method attains positive returns in the box-pushing problem). For all other noise levels < 1.0, the distinct-phase framework provides agents with synchronised messages for at least some fraction of the time; this advantage would however be lost in our experimental setting.

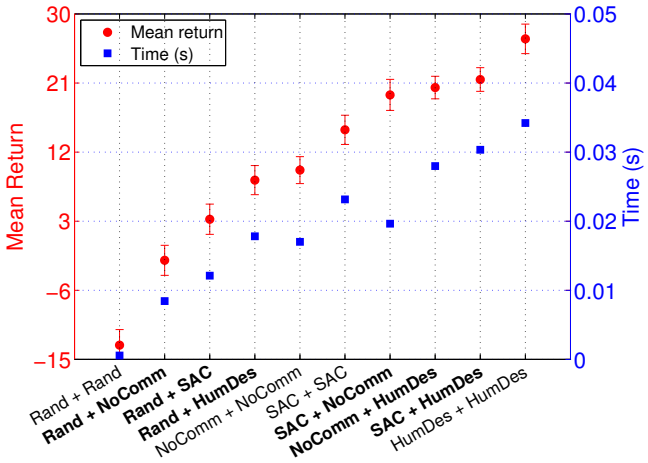
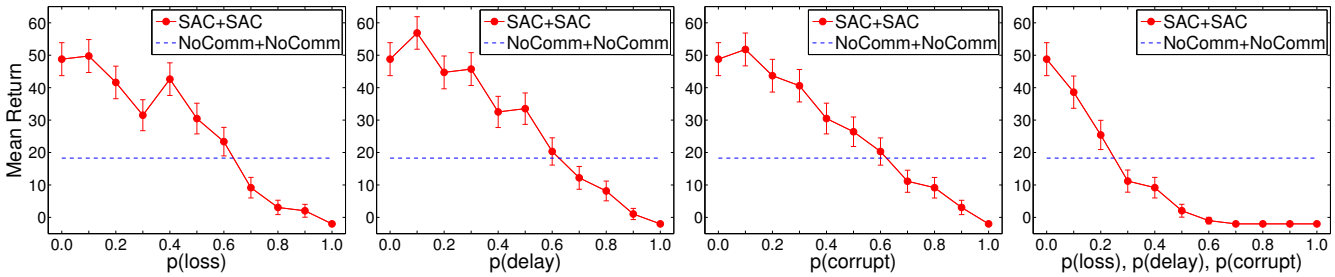


Figure 3: Box pushing domain - results for heterogeneous agent teams. The results are sorted in order of increasing mean return. *Boldface labels:* teams with different agents. *Non-boldface labels:* teams with the same agents. Communication noise: $p(loss) = p(delay) = p(corrupt) = 0.1$. See text for description of the different algorithms.

method can achieve robust collaboration even with agents who do not use any communication.

Multi-robot kitchen domain

The multi-robot kitchen domain is an extension of a single-agent problem described by Petrick, Geib, and Steedman (2009). In the multi-agent variant, two bi-manual robots are tasked with transporting a tray between two different kitchen locations. The kitchen has five locations (*sideboard*, *stove*, *fridge*, *dishwasher*, *cupboard*), and each robot can *move* between them. A location can be *open*ed or *close*d by a robot's left or right hands. The tray can be *grasp*ed or *put_down* at a location, or *transport*ed between locations; these actions are joint and fail if not simultaneously executed by both robots. If a joint action fails at the start location, the tray is dropped and needs to be *place*d_*upright* by one robot; if it fails at any



(a) Message losses (b) Message delays (c) Message corruptions (d) All types of noise combined

Figure 4: Kitchen domain - results for different types of communication noise. See caption of Figure 2 for further explanations.

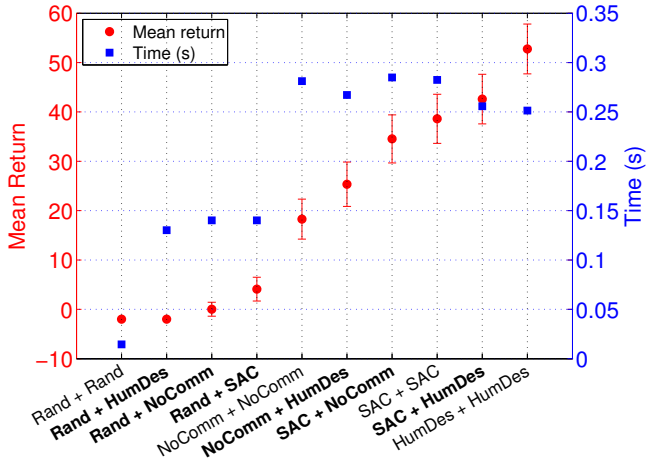


Figure 5: Kitchen domain - results for heterogeneous agent teams. See caption of Figure 3 for further explanations.

other location, the tray is moved back to the start. The tray and teammate are visible only when in the same location as the planning robot. The reward is +100 for successfully taking the tray to the goal, and -0.1 for every other step.

When aggregating all possible object/location/gripper combinations, the kitchen domain has a total of 175 actions per agent. This represents a considerably larger problem space than the box-pushing domain, which impacts both planning and communication (since, as previously discussed, the number of possible message misinterpretations grows with the action space). Additionally, several *distinct* joint actions are now needed to achieve the goal, i.e. *grasp*, *transport*, and *put_down* (as opposed to just *moving* the large box). Another distinguishing feature is that no goal can now be attained by a single agent (as with small boxes previously), so robots *must* collaborate to get a positive reward.

The higher difficulty of the kitchen domain is illustrated in Figure 4, where the **SAC** algorithm now performs worse than **NoComm** in some of the more restricting noise cases. This is particularly evident in Figure 4(d), where the performance decline is more rapid. Nevertheless, even in this challenging problem, **SAC** outperforms **NoComm** under limited communication noise, while exhibiting comparable sensitivity to the different noise types (with 0.6 being the cut-off probability threshold in Figures 4(a), 4(b), and 4(c)).

The **SAC** agent also maintains its ability to achieve superior collaboration with heterogeneous agents than **NoComm** (Figure 5). When comparing with the box-pushing problem results, **SAC** now also outperforms **HumDes** when paired with **Rand** and **NoComm**, thus indicating better adaptation to unknown teammates in this more challenging domain. Furthermore, the **SAC** approach demonstrates comparable efficiency to the other algorithms, as indicated by the recorded computation times.

Conclusions

In this paper, we introduce a novel approach to collaboration in partially observable domains, which is based on the simultaneous execution and exchange of actions between teammates. We extend a state-of-the-art, single-agent Monte-Carlo planner to support egocentric reasoning in multi-agent systems, where communicated messages are used to bias the sampling process and learn policies through factored value updates. Thus, unlike many existing methods that rely heavily on observation and belief synchronisation within a team, our work assumes a looser coupling between planning and communication phases. As demonstrated by our results, our approach outperforms a non-communicative variant in a benchmark domain under varying noise types (message losses, delays, corruptions), while achieving robust collaboration with unknown teammates even in a larger and more complex collaborative planning problem.

We are currently working on integrating communication-based planning with reinforcement learning techniques that actively model the rewards of interacting agents. Our goal is to develop fast, robust decentralised planning algorithms that can be applied to challenging problems with varying task specifications and team compositions. In particular, we are interested in collaborative human-robot interaction applications requiring heterogeneous agents to work (and communicate) in teams towards a common goal, under limited resources and tight coordination constraints.

Acknowledgment

This work has been funded by the European Commission through the EU Cognitive Systems and Robotics project Xperience (FP7-ICT-270273).

References

Barrett, S.; Agmon, N.; Hazon, N.; Kraus, S.; and Stone, P. 2013a. Communicating with Unknown Teammates. In *AAMAS Adaptive Learning Agents (ALA) Workshop*.

Barrett, S.; Stone, P.; Kraus, S.; and Rosenfeld, A. 2013b. Teamwork with Limited Knowledge of Teammates. In *AAAI*.

Becker, R.; Lesser, V.; and Zilberstein, S. 2005. Analyzing Myopic Approaches for Multi-Agent Communication. In *Proceedings of Intelligent Agent Technology*, 550–557.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Math. Oper. Res.* 27(4):819–840.

Coles, A. J.; Coles, A.; Olaya, A. G.; Celorrio, S. J.; López, C. L.; Sanner, S.; and Yoon, S. 2012. A Survey of the Seventh International Planning Competition. *AI Magazine* 33(1).

Gelly, S.; Kocsis, L.; Schoenauer, M.; Sebag, M.; Silver, D.; Szepesvári, C.; and Teytaud, O. 2012. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Commun. ACM* 55(3):106–113.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* 101(1-2):99–134.

Kocsis, L., and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning (ECML)*, 282–293.

Oliehoek, F. A., and Spaan, M. T. J. 2012. Tree-Based Solution Methods for Multiagent POMDPs with Delayed Communication. In *AAAI*.

Oliehoek, F. A.; Spaan, M. T. J.; and Vlassis, N. 2007. Dec-POMDPs with delayed communication. In *AAMAS Workshop on Multi-agent Sequential Decision Making in Uncertain Domains*.

Papadimitriou, C., and Tsitsiklis, J. N. 1987. The Complexity of Markov Decision Processes. *Math. Oper. Res.* 12(3):441–450.

Petrick, R.; Geib, C.; and Steedman, M. 2009. Integrating Low-Level Robot/Vision with High-Level Planning and Sensing in PACO-PLUS. *Technical Report, PACO-PLUS Project Deliverable 4.3.5* (available at `http://www.paco-plus.org`).

Pynadath, D. V., and Tambe, M. 2002. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *J. Artif. Intell. Res. (JAIR)* 16:389–423.

Roth, M.; Simmons, R.; and Veloso, M. 2005. Reasoning About Joint Beliefs for Execution-time Communication Decisions. In *AAMAS*, 786–793.

Seuken, S., and Zilberstein, S. 2007. Memory-Bounded Dynamic Programming for DEC-POMDPs. In *IJCAI*.

Silver, D., and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In *NIPS*, 2164–2172.

Spaan, M. T. J.; Oliehoek, F. A.; and Vlassis, N. A. 2008. Multiagent Planning Under Uncertainty with Stochastic Communication Delays. In *ICAPS*, volume 8, 338–345.

Stone, P.; Kaminka, G. A.; Kraus, S.; and Rosenschein, J. S. 2010. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *AAAI*.

Wu, F.; Zilberstein, S.; and Chen, X. 2011a. Online Planning for Ad Hoc Autonomous Agent Teams. In *IJCAI*, 439–445.

Wu, F.; Zilberstein, S.; and Chen, X. 2011b. Online Planning for Multi-Agent Systems with Bounded Communication. *Artificial Intelligence* 175(2):487–511.

Zhang, C., and Lesser, V. R. 2013. Coordinating multi-agent reinforcement learning with limited communication. In *AAMAS*, 1101–1108.

Closed-form Solutions to a Subclass of Continuous Stochastic Games via Symbolic Dynamic Programming

Shamin Kinathil
ANU and NICTA
Canberra, ACT, Australia
shamin.kinathil@anu.edu.au

Scott Sanner
NICTA and ANU
Canberra, ACT, Australia
ssanner@nicta.com.au

Nicolás Della Penna
ANU and NICTA
Canberra, ACT, Australia
nicolas.della-penna@anu.edu.au

Abstract

Zero-sum stochastic games provide a formalism to study competitive sequential interactions between two agents with diametrically opposing goals and evolving state. A solution to such games in the case of a discrete state, was presented in (Littman, 1994). The continuous state case, however, remains unsolved. In many instances this requires nonlinear parameterised optimisation, a problem for which closed-form solutions are generally unavailable. We characterise a subclass of continuous stochastic games and present a novel symbolic dynamic programming method which enables these problems to be reduced to a parameterised linear program, for which an exact closed-form solution exists. This solution is empirically applied to compute *exact* solutions to a variety of *continuous state* zero-sum stochastic games.

Introduction

Modelling competitive sequential interactions between agents has important applications within economics and decision-making. Stochastic games (Shapley, 1953) provide a convenient framework to model sequential interactions between non-cooperative agents. In zero-sum stochastic games, participating agents have diametrically opposing goals. A reinforcement learning solution to zero-sum stochastic games with discrete states was presented in (Littman, 1994). Closed-form solutions for the continuous state case remain unknown, despite the general importance of this formalism — zero-sum continuous state stochastic games provide a convenient framework with which to model robust sequential optimisation in adversarial settings including many important financial and economic domains such as option valuation on derivative markets.

The difficulty of solving zero-sum continuous state stochastic games arises from the need to calculate a Nash equilibrium for every state, of which there are infinitely many. In this paper we characterise a subclass of continuous state stochastic games for which we can calculate exact solutions for arbitrary horizons via closed-form symbolic dynamic programming (SDP) (Boutilier *et al.*, 2001) in continuous domains (Sanner *et al.*, 2011; Zamani & Sanner, 2012).

We begin by presenting Markov Decision Processes (MDPs) (Howard, 1960) and value iteration (Bellman, 1957), a commonly used dynamic programming solution for MDPs. We then describe both discrete and continuous state zero-sum stochastic games as game-theoretic generalisations of the MDP framework. Following this we show how symbolic dynamic programming can be used to calculate the first known exact solution to a particular subclass of zero-sum continuous stochastic games. We conclude by calculating exact solutions to continuous state matching pennies and a binary option valuation game.

In this paper we make the following key contributions:

- We characterise a subclass of zero-sum continuous stochastic games with restricted reward and transition functions that can be solved exactly via parameterised linear optimisation.
- We provide an algorithm that solves this subclass of stochastic games exactly and optimally using symbolic dynamic programming for arbitrary horizons.

Markov Decision Processes

A Markov Decision Process (MDP) (Howard, 1960) is defined by the tuple $\langle S, A, T, R, h, \gamma \rangle$. S and A specify a finite set of states and actions, respectively. T is a transition function $T : S \times A \to S$ which defines the effect of an action on the state. R is the reward function $R : S \times A \to \mathbb{R}$ which encodes the preferences of the agent. The horizon h represents the number of decision steps until termination and the discount factor $\gamma \in [0, 1)$ is used to discount future rewards. In general, an agent's objective is to find a policy, $\pi : S \to A$, which maximises the expected sum of discounted rewards over horizon h.

Value iteration (VI) (Bellman, 1957) is a general dynamic programming algorithm used to solve MDPs. VI is based on the set of Bellman equations, which mathematically express the optimal solution of an MDP. They provide a recursive expansion to compute: (1) $V^*(s)$, the expected value of following the optimal policy in state s; and (2) $Q^*(s, a)$, which is the expected quality of taking a in state s, then following the optimal policy. The key idea of the algorithm is to successively approximate $V^*(s)$ and $Q^*(s, a)$ by $V^h(s)$ and $Q^h(s, a)$, respectively, at each horizon h. These two functions satisfy the following recursive relationship:

$$Q^h(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \cdot V^{h-1}(s') \quad (1)$$

$$V^h(s) = \max_{a \in A} \left\{ Q^h(s,a) \right\} \quad (2)$$

The algorithm is executed by first initialising $V^0(s)$ to zero or the terminal reward. Then for each h, $V^h(s)$ is calculated from $V^{h-1}(s)$ via Equations (1) and (2), until the intended h-stage-to-go value function is computed. Value iteration converges linearly in the number of iterations to the true values of $Q^*(s,a)$ and $V^*(s)$ (Bertsekas, 1987).

MDPs can be used to model multiagent systems under the assumption that other agents are part of the environment and have fixed behaviour. As a result, they ignore the difference between responsive agents and a passive environment (Hu & Wellman, 1998). In the next section we present a game theoretic framework which generalises MDPs to situations with two or more agents.

Zero-sum Discrete Stochastic Games

Discrete state stochastic games (DSGs) are formally defined by the tuple $\langle S, A_1, \ldots, A_n, T, R_1, \ldots, R_n, h, \gamma \rangle$. S is a set of discrete states and A_i is the action set available to agent i. T is a transition function $T : S \times A_1 \times \ldots \times A_n \to \Delta(S)$, where $\Delta(S)$ is the set of probability distributions over the state space S. The reward function $R_i : S \times A_1 \times \ldots \times A_n \to \mathbb{R}$ encodes the individual preferences of agent i. The horizon h represents the number of decision steps until termination and the discount factor $\gamma \in [0,1)$ is used to discount future rewards. In general, an agent's objective is to find a policy, $\pi_i : S \to \sigma_i(A_i)$ which maximises the expected sum of discounted rewards over horizon h. Here $\sigma_i(A_i)$ specifies probability distributions over action set A_i. The optimal policy in a DSG may be stochastic, that is, it may define a mixed strategy for each state.

Zero-sum DSGs are a type of DSG involving two agents with diametrically opposing goals. Under these conditions the reward structure for the game can be represented by a single reward function since an agents reward function is simply the opposite of their opponent's. The objective of each agent is to maximise its expected discounted future rewards in the minimax sense. That is, each agent views its opponent as acting to minimise the agent's reward. Zero-sum DSGs can be solved using a technique analogous to value iteration for MDPs (Littman, 1994). The value function, $V^h(s)$, in this setting can be defined as:

$$V^h(s) = \max_{m \in \sigma_1(A_1)} \min_{o \in \sigma_2(A_2)} \sum_{a_1 \in A_1} \sum_{a_2 \in A_2} Q^h(s,a_1,a_2) \cdot m_{a_1} \cdot o_{a_2} \quad (3)$$

where $m \in \mathbb{R}^{|A_1|}$ and $o \in \mathbb{R}^{|A_2|}$ are mixed (stochastic) strategies from $\sigma_1(A_1)$ and $\sigma_2(A_2)$, respectively. $Q^h(s,a_1,a_2)$, the quality of taking action a_1 against action a_2 in state s, is given by:

$$Q^h(s,a_1,a_2) = R(s,a_1,a_2) + \gamma \cdot \sum_{s' \in S} T(s,a_1,a_2,s') \cdot V^{h-1}(s'). \quad (4)$$

Equation (3) can be further simplified by noting that since the min operation is "inside" the max, the minimum is achieved for a deterministic action choice. This observation leads to the following form:

$$V^h(s) = \max_{m \in \sigma_1(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q^h(s,a_1,a_2) \cdot m_{a_1}. \quad (5)$$

Together Equations (4) and (5) define a recursive method to calculate the optimal solution to zero-sum DSGs. The policy for the opponent can be calculated by applying symmetric reasoning and the Minimax theorem (Neumann, 1928).

Solution Techniques

Zero-sum DSGs can be solved via discrete linear optimisation. The value function in Equation (5) can be reformulated as a linear program through the following steps:

1. Define $V^h(s)$ to be the value of the inner minimisation term in Equation (5). This leads to the following linear program for a known state s:

$$\begin{aligned} &\text{maximise } V^h(s) \\ &\text{subject to} \\ &V^h(s) = \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q^h(s,a_1,a_2) \cdot m_{a_1} \quad (6a) \\ &\sum_{a_1 \in A_1} m_{a_1} = 1;\ m_{a_1} \geq 0 \qquad \forall a_1 \in A_1 \end{aligned}$$

2. Replace the equality (=) in constraint (6a) with $\leq$ by observing that the maximisation of $V^h(s)$ effectively pushes the $\leq$ condition to the = case. This gives:

$$\begin{aligned} &\text{maximise } V^h(s) \\ &\text{subject to} \\ &V^h(s) \leq \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q^h(s,a_1,a_2) \cdot m_{a_1} \quad (7a) \\ &\sum_{a_1 \in A_1} m_{a_1} = 1;\ m_{a_1} \geq 0 \qquad \forall a_1 \in A_1 \end{aligned}$$

3. Remove the minimisation operator in constraint (7a) by noting that the minimum of a set is less than or equal to the minimum of all elements in the set. This leads to the final form of the discrete linear optimisation problem:

$$\begin{aligned} &\text{maximise } V^h(s) \\ &\text{subject to} \\ &V^h(s) \leq \sum_{a_1 \in A_1} Q^h(s,a_1,a_2) \cdot m_{a_1} \ \forall a_2 \in A_2 \\ &\sum_{a_1 \in A_1} m_{a_1} = 1;\ m_{a_1} \geq 0 \qquad \forall a_1 \in A_1 \end{aligned}$$

We can now use existing linear programming solvers to compute the optimal solution to this linear program for each $s \in S$ at a given horizon h.

The linear program used to solve zero-sum DSGs cannot be used once the state is continuous since there are infinitely many states. The key innovation of this paper is in showing that continuous state zero-sum games can still be solved through the use of symbolic dynamic programming.

Zero-sum Continuous Stochastic Games

Continuous state stochastic games (CSGs) are formally defined by the tuple $\langle \vec{x}, A_1, \ldots, A_n, T, R_1, \ldots, R_n, h, \gamma \rangle$. In CSGs states are represented by vectors of continuous variables, $\vec{x} = (x_1, \ldots, x_m)$, where $x_i \in \mathbb{R}$. The other components of the tuple are as previously defined for discrete stochastic games in the preceding section.

Zero-sum CSGs impose the same restrictions on the number of agents and the reward structure as their discrete state counterparts. The optimal solution to zero-sum CSGs can be calculated via the following recursive functions:

$$Q^h(\vec{x}, a_1, a_2) = R(\vec{x}, a_1, a_2) \quad + \gamma \cdot \int T(\vec{x}, a_1, a_2, \vec{x}') \cdot V^{h-1}(\vec{x}')\, d\vec{x}' \quad (8)$$

$$V^h(\vec{x}) = \max_{m \in \sigma(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q^h(\vec{x}, a_1, a_2) \cdot m_{a_1} \quad (9)$$

We can derive Equation (8) from Equation (4) by replacing s, s' and the $\sum$ operator with their continuous state counterparts, $\vec{x}$, $\vec{x}'$ and $\int$, respectively.

Solution Techniques

Zero-sum CSGs can be solved using a technique analogous to that presented in Section . Namely, the value function in Equation (9) can be reformulated as the following continuous optimisation problem:

maximise $V^h(\vec{x})$

subject to

$$V^h(\vec{x}) \leq \sum_{a_1 \in A_1} Q^h(\vec{x}, a_1, a_2) \cdot m_{a_1} \; \forall a_2 \in A_2 \quad (10a)$$

$$\sum_{a_1 \in A_1} m_{a_1} = 1;\; m_{a_1} \geq 0 \qquad \forall a_1 \in A_1$$

This optimisation problem cannot be easily solved using existing techniques due to two factors: (1) there are infinitely many states in $\vec{x}$; and (2) constraint (10a) is nonlinear in $\vec{x}$ and m_{a_1} for general representations of $Q^h(\vec{x}, a_1, a_2)$. To further illustration the second limitation consider $Q^h(\vec{x}, a_1, a_2)$ in the form of a linear function in x for some a_1 and a_2:

$$Q^h(\vec{x}, a_1, a_2) = \sum_j c_j \cdot x_j \quad (11)$$

Substituting Equation (11) into constraint (10a) yields:

$$V^h(\vec{x}) \leq \sum_{a_1 \in A_1} m_{a_1} \sum_j c_j \cdot x_j \qquad \forall a_2 \in A_2. \quad (12)$$

It is clear from Equation (12) that a linear representation of $Q^h(\vec{x}, a_1, a_2)$ leads to a nonlinear constraint where m_{a_1} must be optimal with respect to the free variable $\vec{x}$ since we need to solve for all states $\vec{x}$. This results in a parameterised nonlinear program, whose optimal solutions are known to be NP-hard (Bennett & Mangasarian, 1993; Petrik & Zilberstein, 2011).

At this point we present the first key insight of this paper: we can transform constraint (10a) from a parameterised nonlinear constraint to a piecewise linear constraint by imposing the following restrictions: (1) restricting the reward function, $R(\vec{x}, a_1, a_2)$, to piecewise constant functions; and (2) restricting the transition function, $T(\vec{x}, a_1, a_2, \vec{x}')$, to piecewise linear functions.

In the next section we show that zero-sum CSGs, with the aforementioned restrictions, can be solved optimally for arbitrary horizons using symbolic dynamic programming.

Symbolic Dynamic Programming

Symbolic dynamic programming (SDP) (Boutilier *et al.*, 2001) is the process of performing dynamic programming via symbolic manipulation. In the following sections we present a brief overview of SDP for zero-sum continuous stochastic games. We refer the reader to (Sanner *et al.*, 2011; Zamani & Sanner, 2012) for more thorough expositions of SDP and its operations.

Case Representation

Symbolic dynamic programming assumes that all symbolic functions can be represented in case statement form (Boutilier *et al.*, 2001) as follows:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases}$$

Here the ϕ_i are logical formulae defined over the state $\vec{x}$ and can include arbitrary logical combinations of boolean variables and linear inequalities over continuous variables. Each ϕ_i is disjoint from the other ϕ_j $(j \neq i)$ and may not exhaustively cover the state space. Hence, f may only be a partial function. In this paper we restrict the f_i to be either linear or constant functions of the state variable $\vec{x}$. We require f to be continuous.

Case Operations

Unary operations on a case statement *f*, such as scalar multiplication $c \cdot f$ where $c \in \mathbb{R}$ or negation $-f$, are applied to each f_i $(1 \leq i \leq k)$.

Binary operations on two case statements are executed in two stages. Firstly, the cross-product of the logical partitions of each case statement is taken, producing paired partitions. Finally, the binary operation is applied to the resulting paired partitions. The "cross-sum" $\oplus$ operation can be performed on two cases in the following manner:

$$\begin{cases}\phi_1 : f_1 \\ \phi_2 : f_2\end{cases} \oplus \begin{cases}\psi_1 : g_1 \\ \psi_2 : g_2\end{cases} = \begin{cases}\phi_1 \wedge \psi_1 : & f_1 + g_1 \\ \phi_1 \wedge \psi_2 : & f_1 + g_2 \\ \phi_2 \wedge \psi_1 : & f_2 + g_1 \\ \phi_2 \wedge \psi_2 : & f_2 + g_2\end{cases}$$

"cross-subtraction" $\ominus$ and "cross-multiplication" $\otimes$ are defined in a similar manner but with the addition operator replaced by the subtraction and multiplication operators, respectively. Some partitions resulting from case operators may be inconsistent and are thus removed.

Minimisation over cases, known as $\mathrm{casemin}$, is defined as:

$$\mathrm{casemin}\left(\begin{cases}\phi_1 : f_1 \\ \phi_2 : f_2\end{cases}, \begin{cases}\psi_1 : g_1 \\ \psi_2 : g_2\end{cases}\right) = \begin{cases}\phi_1 \wedge \psi_1 \wedge f_1 < g_1 : f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \geq g_1 : g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 < g_2 : f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \geq g_2 : g_2 \\ \vdots \qquad\qquad\qquad\quad \vdots\end{cases}$$

$\mathrm{casemin}$ preserves the linearity of the constraints and the constant or linear nature of the f_i and g_i. If the f_i or g_i are quadratic then the expressions $f_i > g_i$ or $f_i \leq g_i$ will be at most univariate quadratic and any such constraint can be linearised into a combination of at most two linear inequalities by completing the square.

The $\mathrm{casemax}$ operator, which calculates symbolic case maximisation, is defined as:

$$\mathrm{casemax}\left(\begin{cases}\phi_1 : f_1 \\ \phi_2 : f_2\end{cases}, \begin{cases}\psi_1 : g_1 \\ \psi_2 : g_2\end{cases}\right) = \begin{cases}\phi_1 \wedge \psi_1 \wedge f_1 > g_1 : f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \leq g_1 : g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 > g_2 : f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \leq g_2 : g_2 \\ \vdots \qquad\qquad\qquad\quad \vdots\end{cases}$$

$\mathrm{casemax}$ preserves the linearity of the constraints and the constant or linear nature of the f_i and g_i.

Other important SDP operations include substitution and continuous maximisation. The substitution operation simply takes a set θ of variables and their substitutions, e.g. $\theta = \{x_1'/(x_1 + x_2), x_2'/x_1^2 \exp(x_2)\}$, where the LHS of the / represents the substitution variable and the RHS of the / represents the expression that should be substituted in its place. We can apply the substitution θ to both non-case functions f_i and case partitions ϕ_i as $f_i\theta$ and $\phi_i\theta$, respectively. Substitution into case statements can therefore be written as:

$$f\theta = \begin{cases}\phi_1\theta : & f_1\theta \\ \vdots & \vdots \\ \phi_k\theta : & f_k\theta\end{cases}$$

Maximisation can be used to calculate $f_1(\vec{x}, y) = \max_y f_2(\vec{x}, y)$ with respect to a continuous parameter y. This is a complex case operation whose explanation is beyond the scope of this paper. We refer the reader to (Zamani & Sanner, 2012) for further elucidation on this operator.

Case statements and their operations are implemented using Extended Algebraic Decision Diagrams (XADDs) (Sanner *et al.*, 2011). XADDs provide a compact data structure with which to maintain compact forms of $Q^h(\vec{x}, a_1, a_2)$ and $V^h(\vec{x})$. XADDs also permit the use of linear constraint feasibility checkers to prune unreachable paths in the XADD.

SDP for Zero-sum Continuous Stochastic Games

In this section, we will show that a *a class of zero-sum continuous stochastic games with a closed-form solution is given by stochastic games with (a) piecewise constant rewards and (b) piecewise linear transition functions*, where SDP can be used to calculate such solutions.

We calculate the optimal solution to zero-sum CSGs by first expressing $R(\vec{x}, a_1, a_2)$, $T(\vec{x}, a_1, a_2, \vec{x}')$, $V^0(\vec{x})$ as case statements. We also express m_{a_1} as a trivial case statement representing an uninstantiated symbolic variable:

$$m_{a_1} = \begin{cases}\top : & m_{a_1}\end{cases}$$

We can then compute the optimal solution via the following SDP equations:

$$Q^h(\vec{x}, a_1, a_2) = R(\vec{x}, a_1, a_2) \quad \oplus \\ \gamma \otimes \int T(\vec{x}, a_1, a_2, \vec{x}') \otimes V^{h-1}(\vec{x}')\, d\vec{x}' \quad (13)$$

$$\tilde{Q}^h(\vec{x}, a_1, a_2) \quad = \quad \sum_{a_1 \in A_1} Q^h(\vec{x}, a_1, a_2) \quad \otimes \quad m_{a_1} \quad (14)$$

$$V^h(\vec{x}) = \max_m \\ \mathrm{casemin}\left(\mathrm{casemin}_{a_2 \in A_2}\left(\tilde{Q}^h(\vec{x}, a_1, a_2), \hat{Q}^h(\vec{x}, a_1, a_2)\right), \mathbb{I}\right) \quad (15)$$

Equations (13) and (15) can be derived from Equations (8) and (9) by replacing all functions by their case statement equivalents and replacing operations such as $+$, $\times$ and min, by their symbolic equivalents, $\oplus$, $\otimes$ and $\mathrm{casemin}$, respectively. Equation (14) calculates a symbolic Q function weighted by the variable m_{a_1} for each a_1. In Equation (15) $\mathrm{casemin}$ is a binary operation as previously defined where the second argument $\hat{Q}^h(\vec{x}, a_1, a_2)$ is simply $\tilde{Q}^h(\vec{x}, a_1, a_2)$ instantiated with a particular a_2. Equation (15) also includes an "indicator" function, $\mathbb{I}$, which serves as a summation constraint $\sum_{a_1 \in A_1} m_{a_1} = 1$. The function $\mathbb{I}$ returns 1 when the conjunction of all constraints on each m_{a_1} are satisfied and $-\infty$, otherwise. That is:

$$\mathbb{I} = \\ \begin{cases}\forall a_1 \in A_1 & [(m_{a_1} \geq 0) \wedge (m_{a_1} \leq 1) \wedge (\sum m_{a_1} = 1)] : 1 \\ \forall a_1 \in A_1 \neg & [(m_{a_1} \geq 0) \wedge (m_{a_1} \leq 1) \wedge (\sum m_{a_1} = 1)] : -\infty\end{cases}$$

The summation constraint is included to ensure that the subsequent $\max$ operation returns legal values for m_{a_1}, since $\max(l, -\infty) = l$.

In Algorithm 1 we present a methodology to calculate the optimal h-stage-to-go value function, $V^h(\vec{x})$, Equations (13) and (15). In the algorithm we notationally specify the

arguments of the Q^h and V^h functions when they are instantiated by an operation. For the base case of $h = 0$, we set $V^0(\vec{x})$, expressed in case statement form, to zero or to the terminal reward. For all $h > 0$ we apply the previously defined SDP operations in the following steps:

1. Prime the Value Function. In line 6 we set up a substitution $\theta = \{x_1/x'_1, \ldots, x_m/x'_m\}$, and obtain $V'^h = V^h\theta$, the "next state".
2. Discount and add Rewards. We assume that the reward function contains primed variables and incorporate it in line 8.
3. Continuous Regression. For the continuous state variables in $\vec{x}$ lines 10 - 11 follow the rules of integration w.r.t. a δ function (Sanner *et al.*, 2011) which yields the following symbolic substitution: $\int f(x'_j) \otimes \delta\left[x'_j - g(\vec{z})\right] dx'_j = f(x'_j)\left\{x'_j/g(\vec{z})\right\}$, where $g(\vec{z})$ is a case statement and $\vec{z}$ does not contain x'_j. The latter operation indicates that any occurrence of x'_j in $f(x'_j)$ is symbolically substituted with the case statement $g(\vec{z})$.
4. Incorporate Agent 1's strategy. At this point we have calculated $Q^h(\vec{x}, a_1, a_2)$, as shown in Equation (13). In lines 13 - 14 we weight the strategy for a specific a_1 by m_{a_1}. We note that m_{a_1} is a case statement representing an uninstantiated symbolic variable.
5. Case Minimisation. In lines 16 - 17 we compute the best case for a_2 as a function of all other variables.
6. Incorporate Constraints. In line 19 we incorporate constraints on the m_{a_1} variable: $\sum_{a_1 \in A_1} m_{a_1} = 1$ and $m_{a_1} \geq 0 \wedge m_{a_1} \leq 1 \quad \forall a_1 \in A_1$. The casemin operator ensures that all case partitions which involve illegal values of m_{a_1} are mapped to $-\infty$.
7. Maximisation. In lines 22 - 23 we compute the best response strategy for every state. We note that this can only be done via symbolic methods since there are infinitely many states. At this point we have calculated $V^h(\vec{x})$ as shown in Equation (15).

It can be proved that all of the SDP operations used in Algorithm 1 are closed form for linear piecewise constant or linear piecewise linear functions (Sanner *et al.*, 2011; Zamani & Sanner, 2012). Given a linear piecewise constant $V^0(\vec{x})$ and that SDP operations are closed form, the resulting $V^{h+1}(\vec{x})$ is also linear piecewise constant. Therefore, by induction $V^{h+1}(\vec{x})$ is linear piecewise constant for all horizons h.

In the next section we demonstrate how SDP can be used to compute exact solutions to a variety of zero-sum continuous stochastic games.

Empirical Results

In this section we evaluate our novel SDP solution technique on two domains: (1) continuous stochastic matching pennies; and (2) continuous binary option valuation. The domain descriptions and results are presented below.

Algorithm 1: CSG-VI(CSG, $H, \mathbb{I}$) $\longrightarrow (V^h)$

```
1  begin
2      V^0 := 0, h := 0
3      while h < H do
4          h := h + 1
5          // Prime the value function
6          Q^h := Prime(V^{h-1})
7          // Discount and add Rewards
8          Q^h := R(x, a_1, a_2, x') ⊕ (γ ⊗ Q^h)
9          // Continuous Regression
10         for all x'_j ∈ Q^h do
11             Q^h := ∫ Q^h ⊗ T(x'_j | a_1, a_2, x) d_{x'_j}
12         // Incorporate Agent 1's strategy
13         for all a_1 ∈ A_1 do
14             Q^h := Q^h ⊕ (Q^h(a_1) ⊗ {⊤ : m_{a_1}})
15         // Case Minimisation
16         for all a_2 ∈ A_2 do
17             Q^h := casemin(Q^h, Q^h(a_2))
18         // Incorporate constraints
19         Q^h := casemin(Q^h, I)
20         // Maximisation
21         V^h = Q^h
22         for all a_1 ∈ A_1 do
23             V^h := max_{m_{a_1}} V^h(m_{a_1})
24         // Terminate if early convergence
25         if V^h = V^{h-1} then
26             break
27     return (V^h)
```

Continuous Stochastic Matching Pennies

Matching pennies is a well known zero-sum game with a mixed Nash Equilibrium (Osborne, 2004). In this paper we extend the standard formulation of the game by incorporating continuous state and sequential decisions while still maintaining the zero-sum nature of the reward.

Domain Description We define continuous stochastic matching pennies as an extensive form game between two players $p \in \{1, 2\}$. The aim of a player is to maximise its expected discounted pay-off at a fixed horizon *H*. Our game is played within the interval $[0, 1]$, two fixed variables $c \in [0, 1)$ and $d \in (0, 1]$ with $(c < d)$, are used to partition the interval into three regions $r \in \{1, 2, 3\}$. Each region is associated with its own zero-sum reward structure. The continuous state variable $x \in [0, 1]$ is used to specify which region the the players are competing within.

At each horizon $(h \leq H)$ each player executes an action $a_p \in \{\text{heads}_p, \text{tails}_p\}$. Player 1 "wins" if both players choose the same action. Otherwise, Player 2 wins. The joint actions of the players affect the state x as follows:

$$P(x'|x, a_1, a_2) =$$

$$\delta \left[x' - \begin{cases} (\text{heads}_1) \wedge (\text{heads}_2) \wedge (x \geq k): & x - k \\ (\text{heads}_1) \wedge (\text{tails}_2) \wedge (x \leq 1): & x + k \\ (\text{tails}_1) \wedge (\text{heads}_2) \wedge (x \geq k): & x + k \\ (\text{tails}_1) \wedge (\text{tails}_2) \wedge (x \leq 1): & x - k \end{cases} \right]$$

The constant $k \in (0, 1)$ is a step size which perturbs the state x. If Player 1 wins, the state moves to the left by k, otherwise it moves to the right by k. The Dirac function $\delta[\cdot]$ ensures that the transitions are valid conditional probability functions that integrate to 1. We define the rewards obtained by Player 1 in region r as:

$$R_1^r = \begin{cases} (\text{heads}_1) \wedge (\text{heads}_2): & \alpha_1^r \\ (\text{heads}_1) \wedge (\text{tails}_2): & \alpha_2^r \\ (\text{tails}_1) \wedge (\text{heads}_2): & \alpha_3^r \\ (\text{tails}_1) \wedge (\text{tails}_2): & \alpha_4^r \end{cases} \tag{16}$$

Here we restrict $\alpha_i^r \in \mathbb{R}$. The rewards obtained by Player 2 in the same region are simply $-R_1^r$. Henceforth, we define rewards from the perspective of Player 1. Given this domain description, we investigate the affects of symmetric and asymmetric reward structures on the continuous stochastic matching pennies game. We define a symmetric reward structure as having $\alpha_1^r = \alpha_4^r$ and $\alpha_2^r = \alpha_3^r$. Under a symmetric reward structure the expected reward in each region is the same for both players. An asymmetric reward structure allows the α_i^r to differ in both sign and magnitude. Under this setting the expected rewards for a player may differ across regions. Tables 1 and 2 show examples of symmetric and asymmetric rewards for the continuous stochastic matching pennies domain.

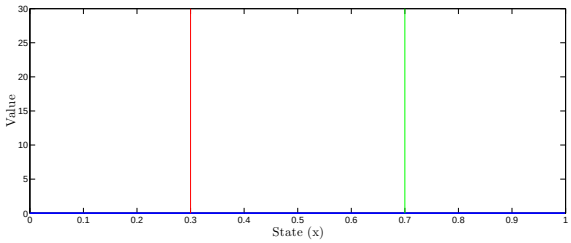
Table 1: Symmetric reward structure for Player 1. Note that symmetric nature of the rewards within each region and the differing rewards between regions.

	Region 1	Region 2	Region 3
$(\text{heads}_1) \wedge (\text{heads}_2)$	10	5	20
$(\text{heads}_1) \wedge (\text{tails}_2)$	-10	-5	-20
$(\text{tails}_1) \wedge (\text{heads}_2)$	-10	-5	-20
$(\text{tails}_1) \wedge (\text{tails}_2)$	10	5	20

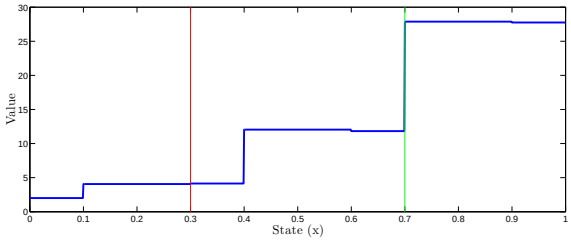
Table 2: Asymmetric reward structure for Player 1. Note that asymmetric nature of the rewards within each region and the differing rewards between regions.

	Region 1	Region 2	Region 3
$(\text{heads}_1) \wedge (\text{heads}_2)$	1	5	7
$(\text{heads}_1) \wedge (\text{tails}_2)$	-3	-5	-2
$(\text{tails}_1) \wedge (\text{heads}_2)$	0	-5	10
$(\text{tails}_1) \wedge (\text{tails}_2)$	2	5	20

Results Figure 1a shows the results of the continuous stochastic matching pennies game using the symmetric reward structure given in Table 1. The results show that the expected reward for Player 1 remains at zero over all 4 horizons, irrespective of the state x. Given the symmetric rewards in each region, both players are indifferent between their pure strategies. Hence, the expected reward for each player is zero in all regions. This corresponds to the well known solution of the matching pennies game where the rewards are symmetric and serves as a proof of concept for our novel solution technique.



(a) Symmetric rewards.



(b) Asymmetric rewards.

Figure 1: Optimal value functions for continuous stochastic matching pennies under (a) symmetric and (b) asymmetric reward structures at horizon 4. Threshold values c and d are highlighted in red and green, respectively. The step size is $k = 0.3$.

The effect of the asymmetric reward structure, given in Table 2, is shown in Figure 1b. From the figure it is clear that Player 1 achieves the highest expected reward in Region 3, followed by Region 2 and finally by Region 1. This is to be expected given the nature of the asymmetric rewards within each region. The results indicate that the two players are no longer indifferent between their pure strategies in each region.

Binary Option Stochastic Game

Binary options are financial instruments which allow an investor to bet on the outcome of a yes/no proposition. The proposition typically relates to whether the price of a particular asset that underlies the option will rise above or fall below a specified amount, known as the strike price, $\kappa \in \mathbb{R}$. When the option reaches maturity the investor receives a fixed pay-off if their bet was correct and nothing otherwise.

Domain Description We analyse the valuation of a binary option as an extensive form zero-sum game between a trader and the market. The aim of the trader is to maximise their expected discounted pay-off at a fixed horizon H through buying and selling options within an adversarial market. The

problem has two state variables: the underlying market value of the option $v \in [0, 100]$ and the trader's inventory of options $i \in \mathbb{N}$.

At each time step the trader can execute one of three actions $a_{trd} \in \{buy_{trd}, sell_{trd}, hold_{trd}\}$, where buy_{trd} refers to a request to buy an option from the market, $sell_{trd}$ refers to a request to sell an option to the market and $hold_{trd}$ is equivalent to taking no action. The market can execute one of two actions: $a_{mkt} \in \{sell_{mkt}, nsell_{mkt}\}$, where $sell_{mkt}$ corresponds to selling an option to the trader and $nsell_{mkt}$ corresponds to not selling to the trader.

The joint actions of the trader and market, a_{trd} and a_{mkt}, respectively, affect both the market value of the option and the trader's inventory. For the sake of simplicity we assume that the market value may increase or decrease by fixed step sizes, $u \in \mathbb{R}$ for an increase and $d \in \mathbb{R}$ for a decrease.

The trader's option inventory dynamics are given by:

$$P(i'|v, i, a_{\text{trd}}, a_{\text{mkt}}) = \delta \begin{bmatrix} i' - \begin{cases} (buy_{trd}) \wedge (sell_{mkt}) : & i+1 \\ (sell_{trd}) \wedge (i > 0) : & i-1 \\ otherwise : & i \end{cases} \end{bmatrix}$$

It should be noted that under this formulation the market will always buy an option from the trader when the trader selects $sell_{trd}$. The market value changes according to:

$$P(v'|v, i, a_{\text{trd}}, a_{\text{mkt}}) = \delta \begin{bmatrix} v' - \begin{cases} (buy_{trd}) \wedge (sell_{mkt}) : & v+u \\ (sell_{trd}) \wedge (i > 0) : & v-d \\ otherwise : & v \end{cases} \end{bmatrix}$$

Assuming that the strike price $\kappa \in [0, 100]$, the rewards obtained by the trader are given by:

$$R_{trader} = \begin{cases} (sell_{trd}) \wedge (i > 0) \wedge (v > \kappa) : & 1 \\ otherwise : & 0 \end{cases}$$

The market's reward is simply the additive inverse of the trader's reward. Hence, the binary option game is zero-sum.

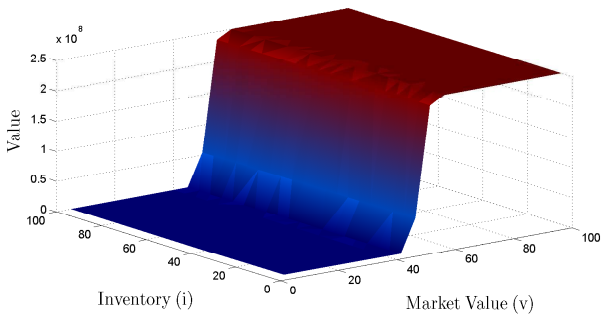


Figure 2: The optimal value function for the binary option game at horizon 20. The strike price is set to $\kappa = 45.0$ and the increment and decrement values are set to $u = 1.0$ and $d = 1.0$, respectively.

Results In Figure 2 we show the optimal value function for the binary option game at horizon 20. The strike price $\kappa = 45.0$ and the increment and decrement values, u and d are both set to 1.0. The value function clearly shows that under this formulation the trader achieves the most reward by selling the option as soon $v > \kappa$. Selling an option triggers the underlying value to decrease, which triggers the trader to buy once the value falls beneath the strike price. This leads to the continual cycling of buying and selling of the option at values close to the strike price κ. In essence the trader behaves like a market maker in that they take both sides of the transaction at values near κ.

Related Work

Solutions to stochastic games have been proposed from within both game theory and reinforcement learning. The first algorithm, game theoretic or otherwise, for finding a solution to a stochastic game was given by Shapley (Shapley, 1953). The algorithm repeatedly calculates a value function $V(s)$ over discrete states which converges to an optimal value function $V^*(s)$, which represents the expected discounted future reward if both players in the game followed the game's Nash equilibrium. Shapley's algorithm is in essence an extension of the value iteration algorithm to stochastic games.

A reinforcement learning based solution to stochastic games was first introduced by (Littman, 1994). Littman's algorithm, Minimax-Q, extends the traditional Q-learning algorithm for MDPs to zero-sum discrete stochastic games. The algorithm converges to the stochastic game's equilibrium solution. Hu and Wellman (Hu & Wellman, 1998) extended Minimax-Q to general-sum games and proved that it converges to a Nash equilibrium under certain restrictive conditions. Although both reinforcement learning based algorithms are able to calculate equilibrium solutions they are limited to discrete state formulations of stochastic games. In this paper we calculate exact solutions to continuous state formulations of stochastic games, under certain restrictions. The Dec-MDP (Bernstein *et al.*, 2002) framework allows for decentralised control within continuous state spaces but is limited to general-sum systems. In this paper we provide the first known exact closed-form solution to a subclass of continuous state zero-sum stochastic games defined by a piecewise constant reward and piecewise linear transition.

Several techniques have been put forward to tackle continuous state spaces in MDPs. Li and Littman (Li & Littman, 2005) describe a method for approximate solutions to continuous state MDPs. In their work, Li and Littman only allow for rectilinearly aligned constraints in their reward and transition functions (not arbitrary linear constraints) and cannot handle general linear transition models without approximation. Our SDP method provides exact solutions without these restrictions, which makes SDP strictly more general. Also, Li and Littman did not consider game-theoretic extensions of their work or the parameterised optimisation problem that these extensions entail.

Symbolic dynamic programming techniques have been previously used to calculate exact solutions to single agent MDPs with both continuous state and actions in a variety of

non-game theoretic domains (Sanner *et al.*, 2011; Zamani & Sanner, 2012). In this paper we build on this work and present the first application of SDP to stochastic games with concurrently acting agents.

Conclusions

We have characterised a subclass of zero-sum continuous stochastic games that can be solved exactly via parameterised linear optimisation. We have also presented a novel symbolic dynamic programming algorithm that can be used to calculated exact solutions to this subclass of stochastic games for arbitrary horizons. The algorithm was used to calculate the first known exact solutions to a variety of experimental continuous domains.

There are a number of avenues for future research. Firstly, it is important to examine more general representations of the reward and transition functions while still guaranteeing exact solutions. Another direction of research lies within improving the scalability of the algorithm by using techniques such as bounded error compression for XADDs (Vianna *et al.*, 2013) or lazy approximation of value functions as piecewise linear XADDs (Li & Littman, 2005). Search based approaches such as RTDP (Barto *et al.*, 1995) and HAO* (Meuleau *et al.*, 2009) are also readily adaptable to SDP. These extensions may then be used to model more complex financial and economic domains. Finally, SDP can be used to calculate exact solutions to general sum stochastic games. The advances made by this work open up a number of potential novel research paths that we believe may help make progress in solving game theoretic domains with continuous state.

References

Barto, Andrew G., Bradtke, Steven J., & Singh, Satinder P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence*, **72**(1-2), 81–138.

Bellman, Richard E. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bennett, Kristin P., & Mangasarian, O. L. 1993. Bilinear Separation of Two Sets in N-space. *Comput. Optim. Appl.*, **2**(3), 207–227.

Bernstein, Daniel S., Givan, Robert, Immerman, Neil, & Zilberstein, Shlomo. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research*, **27**(4), 819–840.

Bertsekas, Dimitri P. 1987. *Dynamic Programming: Deterministic and Stochastic Models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Boutilier, Craig, Reiter, Ray, & Price, Bob. 2001. Symbolic Dynamic Programming for First-order MDPs. *Pages 690 – 697 of: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*. IJCAI, vol. 1.

Howard, Ronald A. 1960. *Dynamic Programming and Markov Processes*. The M.I.T. Press.

Hu, Junling, & Wellman, Michael P. 1998. Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm. *Pages 242–250 of: Proceedings of the Fifteenth International Conference on Machine Learning*. ICML. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Li, Lihong, & Littman, Michael L. 2005. Lazy Approximation for Solving Continuous Finite-horizon MDPs. *Pages 1175–1180 of: Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*. AAAI. AAAI Press.

Littman, Michael L. 1994. Markov Games as a Framework for Multi-Agent Reinforcement Learning. *Pages 157–163 of: Proceedings of the 11th International Conference on Machine Learning Machine Learning*. ICML. San Francisco, California, USA: Morgan Kaufmann Publishers Inc.

Meuleau, Nicolas, Benazera, Emmanuel, Brafman, Ronen I., Hansen, Eric A., & Mausam. 2009. A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains. *Journal of Artificial Intelligence Research*, **34**, 27–59.

Neumann, J. 1928. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, **100**(1), 295–320.

Osborne, Martin J. 2004. *An introduction to game theory*. New York: Oxford University Press.

Petrik, Marek, & Zilberstein, Shlomo. 2011. Robust Approximate Bilinear Programming for Value Function Approximation. *Journal of Machine Learning Research*, **12**, 3027–3063.

Sanner, Scott, Delgado, Karina, & Nunes de Barros, Leliane. 2011. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. *Pages 1–10 of: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*. UAI.

Shapley, L. S. 1953. Stochastic Games. *Proceedings of the National Academy of Sciences*, **39**(10), 1095–1100.

Vianna, Luis Gustavo Rocha, Sanner, Scott, & Nunes de Barros, Leliane. 2013. Bounded Approximate Symbolic Dynamic Programming for Hybrid MDPs. *Pages 1–9 of: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI2013)*. UAI.

Zamani, Zahra, & Sanner, Scott. 2012. Symbolic Dynamic Programming for Continuous State and Action MDPs. *Pages 1–7 of: Proceedings of the 26th Conference on Artificial Intelligence (AAAI-12)*. AAAI.

Beliefs in Multiagent Planning: From One Agent to Many

Filippos Kominis
Universitat Pompeu Fabra
08018 Barcelona, Spain

Hector Geffner
ICREA & Universitat Pompeu Fabra
08018 Barcelona, Spain

Abstract

Single-agent planning in partially observable settings is a well understood problem and existing planners can represent and solve a wide variety of meaningful instances. In the most common formulation, the problem is cast as a non-deterministic search problem in belief space where beliefs are sets of states that the agent regards as possible. In this work, we build on the methods developed for representing beliefs in single-agent planning to introduce a simple but expressive formulation for handling beliefs in multi-agent settings. The resulting formulation deals with multiple agents that can act on the world (physical or ontic actions), and can sense either the state of the world (truth of objective formulas) or the mental state of other agents (truth of epistemic formulas). The formulation captures and defines a fragment of dynamic epistemic logics that is simple and expressive but which does not involve event models or product updates, and has the same complexity of belief tracking in the single agent setting and can benefit from the use of similar techniques. We show indeed that the problem of *computing multiagent linear plans* can be actually compiled into a *classical planning problem* using the techniques that have been developed for compiling conformant and contingent problems in the single agent setting.

Introduction

Single-agent planning in partially observable settings is a well understood problem and existing planners can represent and solve a wide variety of meaningful instances. In the most common formulation, single-agent planning in partially observable environments is cast as a non-deterministic search problem in belief space where the beliefs are sets of states that the agent regards as possible (Bonet and Geffner 2000). The work in partially observable or contingent planning has been focused on ways for representing beliefs and selecting actions (Bertoli et al. 2001; Brafman and Hoffmann 2004; Albore, Palacios, and Geffner 2009; To, Pontelli, and Son 2011; Brafman and Shani 2012a).

Current approaches for representing beliefs in multiagent dynamic settings, on the other hand, are based on Kripke structures (Fagin et al. 1995). Multiagent Kripke structures are triplets defined by a set of worlds, accessibility relations among the worlds for each of the agents, and truth valuations that define the propositions that are true in each world. While a truth valuation determines the *objective* formulas that are true in a world, the accessibility relation among worlds provides the truth conditions for *epistemic* formulas.

Dynamic epistemic logics extend epistemic logics with the ability to deal with change (van Ditmarsch, van der Hoek, and Kooi 2007; van Ditmarsch and Kooi 2008; Van Benthem 2011). The standard approach relies on *event models* and *product updates* by which both the agent beliefs and the events are represented by Kripke structures, and the resulting beliefs are captured by a suitable cross product of the two (Baltag, Moss, and Solecki 1998; Baltag and Moss 2004). Syntactically, axiomatizations have been developed to capture the valid inferences in such a setting, and a number of approaches have been developed to facilitate modeling and inference (Baral et al. 2012; Herzig, Lang, and Marquis 2005). A simple form of planning, however, where an event sequence is sought to achieve a given goal formula, has been shown to be undecidable in dynamic epistemic logic (Aucher and Bolander 2013), while decidable subsets have been identified as well (Löwe, Pacuit, and Witzel 2011).

In this work, we build on the methods developed for representing beliefs in single-agent planning to introduce a simple but expressive formulation for handling beliefs in multi-agent settings. The resulting formulation deals with multiple agents that can act on the world (physical or ontic actions), and can sense either the state of the world (truth of objective formulas) or the mental state of other agents (truth of epistemic formulas). The formulation captures and defines a fragment of dynamic epistemic logics that is simple and expressive, but which does not involve event models or product updates, and has the same complexity of belief tracking in the single agent setting and can benefit from the use of similar techniques. We show indeed that the problem of computing *linear multiagent plans* (Bolander and Andersen 2011) can be actually compiled into a *classical planning problem*, using the techniques that have been developed for compiling conformant and contingent problems in the single agent setting (Palacios and Geffner 2009; Brafman and Shani 2012b).

The proposed formulation exploits certain conventions and restrictions. First, while the agents can have private in-

formation as they have private sensors, they are all assumed to start with a *common initial belief* on the set of worlds that are possible. Second, the effects of physical actions on the world are assumed to be *deterministic*. And third, the *sequence of events* (physical actions, sensing events, and public announcements) that can change the state of the world or the knowledge state of the agents, is *public* to all the agents. In the formulation it is crucial to distinguish between the event of sensing the truth value of an objective or epistemic formula, and the agent coming to know that the formula is true or false. While the sensing event is public, as when all agents know the sensor capabilities of the other agents, the actual information provided by these sensors is private. For example, in the muddy children problem (Fagin et al. 1995), every child i is assumed to be capable of sensing the truth value of the atoms m_j encoding whether child j is muddy for $j \neq i$, and every child knows that. Yet this doesn't mean that children have access to the truth values revealed by the sensors that are not their own.

The rest of the paper is organized as follows. We start with a well known example and introduce the modeling language, the belief representation, and the (linear) planning problem. We then analyze other examples, discuss the relation to dynamic epistemic logic, and formulate the compilation of the linear multiagent planning problem into classical planning.

Example

Before proceeding with the details of the formulation, it'll be useful to consider how a familiar example, the Muddy Children Puzzle will be represented (Fagin et al. 1995). We consider three agents $A = \{a, b, c\}$ and atoms m_x, for $x \in A$, each representing that child x is muddy. The states of the problem are the possible truth valuation over these three atoms, and the common initial belief state b_I is given by the set of all such states (8 in total). Consider then the sequence of events σ given by:

$$\begin{array}{r}\textbf{update}(m_a \vee m_b \vee m_c), [\textbf{sense}(a, [m_b, m_c]),\\ \textbf{sense}(b, [m_a, m_c]), \textbf{sense}(c, [m_a, m_b])]\ ,\end{array} \quad (1)$$

that includes the public announcement made by the father, followed by each agent sensing in parallel whether each of the other children is muddy or not. The event $\textbf{sense}(a, \phi)$ expresses that agent a senses the truth value of formula ϕ. Variations of these events, expressed as $\textbf{sense}(a, [\phi_1, \ldots, \phi_n])$, $\textbf{sense}([a, b], [\phi_1, \ldots, \phi_n])$, and $\textbf{sense}([\phi_1, \ldots, \phi_n])$ represent that agent a senses the truth value of each of the formulas ϕ_i, $i = n$, in parallel, that both a and b sense such truth values in parallel, and that all agents sense them in parallel. In addition, groups of sensing events can be enclosed in brackets as in (1), meaning that the events are in parallel.

A possible query after the sequence of events σ may be whether any of the agents will know that he is muddy if the world is such that there is just one muddy child. This query would amount to testing the formula

$$m_a \oplus m_b \oplus m_c \Rightarrow K_a m_a \vee K_b m_b \vee K_c m_c \quad (2)$$

in (the Kripke structure associated with) the situation resulting from the common initial belief state b_I and the event sequence σ. In this formula, '$\oplus$' stands for the "exclusive or"; $p \oplus q$ thus being an abbreviation of $(p \vee q) \wedge \neg(p \wedge q)$. The answer to this query will be positive. On the other hand, the answer to the query:

$$\neg(m_a \oplus m_b \oplus m_c) \Rightarrow K_a m_a \vee K_b m_b \vee K_c m_c \quad (3)$$

will be negative, as when there is more than one muddy child, no child will know that he is muddy from the announcement made by the father and the information gathered from his physical sensors alone. It can be shown, however, that if the event sequence σ is extended with the following parallel sensing event:

$$[\textbf{sense}(K_a m_a), \textbf{sense}(K_b m_b), \textbf{sense}(K_c m_c)] \quad (4)$$

where all agents learn whether each of the agents knows that he is muddy, a formula like

$$m_a \wedge m_b \wedge \neg m_c \Rightarrow K_a m_a \quad (5)$$

will become true, as in the world where a and b are muddy and c is not, the sensing captured by (4) would result in a learning that b does not know that b is muddy ($K_a \neg K_b m_b$), while in the other world that is possible to a, where a is not muddy, a would come to learn the opposite; namely that b knows that b is actually muddy ($K_a K_b m_b$).

Language

We consider planning problems $P = \langle A, F, I, O, N, U, G\rangle$ where A is a set of agent names, F is a set of atoms, I is the initial situation, O is a set of *physical actions*, N is a set of *sensing actions*, U is set of *public updates*, and G is the goal. A *plan* for P, as in classical planning, is a *sequence of actions* for achieving the goal G from the initial situation described by I. The main differences to classical planning result from the uncertainty in the initial situation, that makes the problem similar to *conformant planning*, and the beliefs of the *multiple agents* involved. In addition the actions may come from any of the sets O, N, or U. If we let S stand for the set of all possible truth-valuations s over the atoms in F and call such valuations *states*, we assume that I is an objective formula over F which denotes a non-empty set of possible initial states b_I. A physical action a in O denotes a *deterministic* state-transition function f_a that maps any state s into a state $s' = f_a(s)$. A (parallel) sensing action in N is a set of expressions of the form $\textbf{sense}[A_k](\phi_k)$, where A_k is a non-empty set of agent names and ϕ_k is an objective or epistemic formula over the atoms F and the knowledge modalities K_i for $i \in A$. The action updates in U are denoted by expressions of the form $\textbf{update}(\phi)$ where ϕ is a formula. Finally, each action a has a precondition $Pre(a)$, which like the goal G are formulas as well. The grammar of these formulas can be expressed as:

$$\phi = p \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi \Rightarrow \phi) \mid K_i \phi$$

where p is an atom in F, and i an agent in A.

We regard *plans* as linear sequences of actions (Bolander and Andersen 2011), and call P a *linear multiagent planning*

problem. While many problems require *non-linear plans*, as it is the case in contingent planning, *linear plans* suffice for a number of non-trivial contexts and provide the basis for more complex forms of plans.

Belief Update and Dynamics

In order to define the belief representation and dynamics, let us represent the event sequences or plans σ over a problem P by sequences of the form $e(0), \ldots, e(t)$, where $e(k)$ is the event from P that occurs at time k. When convenient, we will assume that the agent names are positive numbers i, $i = 1, \ldots, m$ for $m = |A|$, or that they can enumerated in this way.

The beliefs of all the agents at time step t, called also the joint belief, will be denoted as $B(t)$, and it is represented by a vector of *conditional beliefs* $B(s,t)$, where s is one of the possible initial states, $s \in b_I$; namely,

$$B(t) = \{B(s,t) \mid s \in b_I \}\ . \tag{6}$$

The conditional beliefs $B(s,t)$ represent the beliefs of all the agents at time t, *under the assumption that the true but hidden initial state is s*. The reason for tagging beliefs with possible initial states is that for a *fixed (hidden) initial state s, the evolution of the beliefs $B(s,t)$ after an arbitrary event sequence is deterministic*.[1] These conditional beliefs $B(s,t)$ are in turn represented by tuples:

$$B(s,t) = \langle v(s,t), r_1(s,t), r_2(s,t), ..., r_m(s,t) \rangle \tag{7}$$

where $v(s,t)$ is the state of the world that results from the initial state s after the event sequence $e(0), \ldots, e(t-1)$, and $r_i(s,t)$ is the set of possible *initial* states $s' \in b_I$ that agent i cannot distinguish at time t from the actual initial state s. Note that s may be the true initial state, and yet the agents may not know about it. Indeed, initially, they only know that if s is the true initial state, it must be part of the initial common belief b_I.

More precisely, the initial beliefs $B(s,t)$ at time $t = 0$ are given by:

$$v(s,t) = s \ \text{ and } \ r_i(s,t) = b_I \tag{8}$$

for all agents i, meaning that under the assumption that the hidden initial state is s and that no events have yet occurred, the actual state is s and the set of possible initial states is b_I.

The belief $B(t+1)$ at time $t+1$ is a function of the belief $B(t)$ and event $e(t)$ at time t:

$$B(t+1) = \mathbf{F}(B(t), e(t)) \tag{9}$$

We express this function by defining how the *type of event* $e(t)$ at time t affects the state $v(s,t+1)$ and the relations $r_i(s,t+1)$ that define the belief $B(t+1)$ at time $t+1$.

Physical Actions: If $e(t) = \mathbf{do}(a)$ for action a denoting a state-transition function f_a, then the current state $v(s,t)$ associated with the hidden initial state s changes according to f_a, but the sets of initial states $r_i(s,t)$ that agent i regards as possible remains unchanged

$$v(s,t+1) = f_a(v(s,t)) \tag{10}$$
$$r_i(s,t+1) = r_i(s,t) \tag{11}$$

where the index i ranges over all the agents in A.

All the other event types affect instead the sets $r_i(s,t+1)$ but not the state $v(s,t+1)$ that is regarded as current given the assumption that s is the true initial hidden state. That is, for the following event types $v(s,t+1) = v(s,t)$.

Sensing: If $e(t) = [\mathbf{sense}[A_1](\phi_1), \ldots, \mathbf{sense}[A_l](\phi_l)]]$ is a sensing action denoting the set of sensing expressions $\mathbf{sense}[A_k](\phi_k)$ done in parallel at time t, the current state given s does not change, but the set of possible initial states compatible with the hidden initial state s for agent i given by $r_i(s,t+1)$ becomes:

$$\{s' \mid s' \in r_i(s,t) \text{ and } B(t), s' \models \phi_k \text{ iff } B(t), s \models \phi_k\} \tag{12}$$

where k ranges over all the indices in $[1,l]$ such that A_k includes agent i. If there are no such indices, $r_i(s,t+1) = r_i(s,t)$. The expression $B(t), s \models \phi$ denotes that ϕ is true in the belief at time t conditional on s being the true hidden state. The truth conditions for these expressions are spelled out below.

Updates: If $e(t) = \mathbf{update}(\phi)$, $r_i(s,t+1)$ is

$$\{s' \mid s' \in r_i(s,t) \text{ and } B(t), s' \models \phi\}\ . \tag{13}$$

The intuition for all these updates is the following. *Physical actions* change the current state of the world according to their state transition function. *Sensing actions* do not change the world but yield information. More specifically, when agent i senses the truth value of formula ϕ at time t, the set of initial states $r_i(s,t+1)$ that he thinks possible under the assumption that the true initial state is s, preserves the states s' in $r_i(s,t)$ that agree with s on the truth value predicted for ϕ at time t. Finally, a *public update* ϕ preserves the possible initial states s' in $r_i(s,t)$ that predict the formula ϕ to be true, and rules out the rest. The conditions under which a possible initial state s predicts that a formula ϕ will be true at time t, and the conditions under which a formula ϕ is true at time t, are made explicit below. Physical, sensing, and update actions are *applicable* at time t only when their preconditions are true at t.

Beliefs and Kripke Structures

A Kripke structure is a tuple $\mathcal{K} = \langle W, R, V \rangle$, where W is the set of worlds, R is a set of binary accessibility relations R_i on W, one for each agent i, and V is a mapping from the worlds w in W into truth valuations $V(w)$. The conditions under which an arbitrary formula ϕ is true in a world w of a Kripke structure $\mathcal{K} = \langle W, R, V \rangle$, written $\mathcal{K}, w \models \phi$, are defined inductively:

[1] In single-agent planning, the idea of tagging beliefs with each of the possible initial states has been used in translations of conformant and partial-observable planning into classical planning (Palacios and Geffner 2009; Brafman and Shani 2012a).

- $\mathcal{K}, w \models p$ for an atom p, if p is true in $V(w)$,
- $\mathcal{K}, w \models \phi \vee \psi$ if $\mathcal{K}, w \models \phi$ or $\mathcal{K}, w \models \psi$,
- $\mathcal{K}, w \models (\phi \Rightarrow \psi)$ if $\mathcal{K}, w \models \phi$ implies $\mathcal{K}, w \models \psi$,
- $\mathcal{K}, w \models K_i\phi$ if $\mathcal{K}, w' \models \phi$ for all w' s.t. $R_i(w, w')$, and
- $\mathcal{K}, w \models \neg\phi$ if $\mathcal{K}, w \not\models \phi$

A formula ϕ is valid in the structure $\mathcal{K}$, written $\mathcal{K} \models \phi$, iff $\mathcal{K}, w \models \phi$ for all worlds w in $\mathcal{K}$. The conditions under which a possible initial state s predicts the truth of a formula ϕ at time t, written $B(t), s \models \phi$, follow from replacing the belief $B(t)$ by the Kripke structure $\mathcal{K}(t) = \langle W^t, R^t, V^t \rangle$ defined by $B(t)$ where

- $W^t = \{s \mid s \in Poss(t)\}$,
- $R^t_i = \{(s, s') \mid \text{ if } s' \in r_i(s, t)\ \}$,
- $V^t(s) = v(s, t)$

In these expressions, $Poss(t)$ stands for initial states that remain possible at time t; i.e,

$$Poss(t) = \cup_{s \in b_I} \cup_{i=1,\ldots,m} r_i(s, t) \tag{14}$$

In other words, the worlds w in the structure $\mathcal{K}(t)$ are the possible initial states $s \in b_I$ of the problem P that have not been ruled out by the updates. The worlds that are accessible from a world s to the agent i are the possible initial states s' that are in $r_i(s, t)$. And finally, the valuation associated to a world s in this structure is the state $v(s, t)$ that deterministically follows from the possible initial state s and the event sequence up to $t - 1$. $B(t), s \models \phi$ is thus true when $\mathcal{K}(t), s \models \phi$ is true, and $B(t) \models \phi$ iff $\mathcal{K}(t) \models \phi$.

It is simple to show that the accessibility relations $R_i(t)$ are reflexive, symmetric, and transitive, meaning that the valid formulas satisfy the axioms of the epistemic logic S5.

Examples

We will show later that a linear multiagent problem P can be translated into a classical planning problem and solved by off-the-shelf planners. Before presenting such a translation, we consider two other examples. As it is common in planning, while the language is propositional, we make use of predicate and actions schemas to characterize the set of (ground) atoms and actions.

Physical and Sensing Actions Combined

Let a, b, and c be three agents in a corridor of four rooms (p_1, p_2, p_3 and p_4 from left to right). The agents can move from a room to a contiguous room, and when agent i communicates (tells) some information, all the agents that are in the same room or in a contiguous room, will hear what was communicated. For example, if agent i expresses in room p_3 his knowledge about q, all agents in rooms p_2, p_3 and p_4 will come to know it. We consider the problem where agent a is initially in room p_1, b in p_2, c in p_3, and a has to find out the truth value of a proposition q and let c know without agent b learning it. To simplify things, we assume that only agent a can move, and that he can learn the value of q in room p_2. A shortest solution to the problem will be for agent a to move right once to p_2 to learn the value of q, to move right twice to p_4, and to communicate the value of q from p_4, so that agent c can listen but agent b can't.

The planning problem is encoded as the tuple $P = \langle A, F, I, O, N, U, G \rangle$ where $A = \{a, b, c\}$, $F = \{q\} \cup \{p(x, i)\}$, $x \in A$, $i \in [1, 4]$, $I = \{p(a, 1), p(b, 2), p(c, 3)\} \cup D$, where D contains the formulas expressing that each agent is in a single room, U is empty, and the goal is

$$G = (K_c q \vee K_c \neg q) \wedge (\neg K_b q \wedge \neg K_b \neg q)\ .$$

The set of physical actions is $O = \{right, left\}$ affecting the location of agent a in the obvious way (the actions have no effects when they'd move the agent away from the four rooms).

The sensing actions in N are two: the first about a learning the value of q when in p_2, the other, about a expressing his knowledge regarding q, which translates into agents b and c learning this when they are close enough to a. The first sensing action is thus **sense**(a, q) with the *precondition* $p(a, 2)$, and the second is

$$\begin{aligned}\textbf{tell}(a, q)\ :\ [&\textbf{sense}(b, \phi_b \Rightarrow K_a q), \textbf{sense}(b, \phi_b \Rightarrow K_a \neg q),\\ &\textbf{sense}(c, \phi_c \Rightarrow K_a q), \textbf{sense}(c, \phi_c \Rightarrow K_a \neg q)]\ ,\end{aligned}$$

where **tell**(a, q) is the abbreviation of the action that we will use, and ϕ_b is the formula expressing that agent b is at distance less than 1 from agent a; namely $\phi_b = \vee_{i,j}[p(a, i) \wedge p(b, j)]$ for i and j in $[1, 4]$ such that $|i - j| \leq 1$. The formula ϕ_c is similar but with c instead of b.

Initially, b_I contains the two states s_1 and s_2 satisfying I, the first where q is true, and the second where it is false. The initial belief at time $t = 0$ is $B(t) = \{B(s_1, t), B(s_2, t)\}$, where $B(s_i, t) = \langle v(s_i, t), r_a(s_i, t), r_b(s_i, t), r_c(s_i, t) \rangle$, $i = 1, 2$, and $r_x(s, t) = b_I$ for $x \in A$ and $s \in b_I$. The shortest plan is

$$\textbf{do}(right), \textbf{sense}(a, q), \textbf{do}(right), \textbf{do}(right), \textbf{tell}(a, q)\ .$$

The first sensing action can be done because its precondition $p(a, 2)$ holds in $B(1)$, and as an effect it removes agent a's uncertainty regarding q making $r_a(s_1, 2) = \{s_1\}$ and $r_a(s_2, 2) = \{s_2\}$. Agent a then knows whether q is true or false, and in principle, he could communicate this from his current location p_2 by performing the action **tell**(a, q) right away. But since the condition ϕ_b is true, b would come to know whether q is true, making the problem goal G unachievable. The effect of the two $right$ actions is to make $p(a, 4)$ true, and all other $p(a, i)$ atoms false, thus making the formula ϕ_b false and the formula ϕ_c true (i.e., agent a is now near c but not near b). The final event in the plan makes the truth value of q known to agent c but not to agent b, thus achieving the goal G. The first part follows because the state $v(s_1, 5)$ where agent a is at p_4 and q is true, makes the formula $\phi_c \Rightarrow K_a q$ sensed by agent c true, while the state $v(s_2, 5)$ makes this formula false, and similarly, the state $v(s_2, 5)$ makes the formula $\phi_c \Rightarrow K_a \neg q$ sensed by agent c true, while the state $v(s_1, 5)$ makes it false. As a result, the state s_2 is not in $r_c(s_1, 5)$, the state s_1 is not in $r_c(s_2, 5)$, both sets become singletons, and hence, the truth value of q becomes known to agent c. The same reasoning does not apply to agent b because the condition ϕ_b is false in the two

states $v(s_1, 5)$ and $v(s_2, 5)$, and hence, both states trivially satisfy the formulas $\phi_b \Rightarrow K_a q$ and $\phi_b \Rightarrow K_a \neg q$ that are sensed by agent b, so that $r_b(s_1, 5)$ and $r_b(s_2, 5)$ remain unchanged, and equal to b_I.

Collaboration through Communication

As another example, we consider a scenario where two agents volunteer information to each other in order to accomplish a task faster that would otherwise be possible without information exchange. It is inspired in the BW4T environment, a proposed testbed for joint activity (Johnson et al. 2009). There is a corridor of four rooms, p_1, p_2, p_3 and p_4 as in the previous example, four blocks $b_1, \ldots, b_4$ that are in some of the rooms, and two agents a and b that can move back and forth along this corridor. Initially, the two agents are in p_2 and do not know where the blocks are (they are not in p_2). When an agent gets into a room, he can see which blocks are in the room if any. The goal of the planning problem is for agent a to know the position of block b_1, and for agent b to know the position of block b_2. A shortest plan for the problem involves six steps: one agent, say a, has to move to p_1, the other agent has to move to p_3, they both must sense which blocks are in these rooms, and then they must exchange the relevant information. At that point, the goal would be achieved whether or not the information exchanged explicitly conveys the location of the target blocks. Indeed, if agent a does not see block b_2 in p_1 and agent b doesn't see this block either at p_3, agent a will then know that block b_2 must be in p_4 once b conveys to a the relevant piece of information; in this case $\neg K_b in(b_2, p_3)$.

The planning problem is $P = \langle A, F, I, O, N, U, G \rangle$, where $A = \{a, b\}$, $F = \{at(x, p_k), in(b_i, p_k)\}$, $x \in A$ $i, k \in [1, 4]$, $I = \{at(a, p_2), at(b, p_2)\} \cup D$, where D contains the formulas expressing that each block has a unique location. The set of updates U is empty, the goal is $G = (\vee_{k=1,4} K_a at(b_1, p_k)) \wedge (\vee_{k=1,4} K_b at(b_2, p_k))$, the actions in O are $right_x$ and $left_x$, for each agent $x \in A$ with the same semantics as in the example above, while the sensing actions are **sense**$(x, [in(b_1, p_k), \ldots, in(b_4, p_k)]$ with precondition $at(x, p_k)$ by which agent $x \in A$ finds out in parallel which blocks b_i, if any, are and are not in p_k, and **sense**$(x, [K_y in(b_i, p_k])$, by which agent y communicates to agent $x \neq y$, whether he knows $in(b_i, p_k)$, $i, k \in [1, 4]$. There are thus four physical actions, eight actions that sense the world, and sixteen communication actions. A shortest plan is:

do$(left_a)$, **do**$(right_b)$, **sense**$(a, [in(b_1, p_1), ..., in(b_4, p_1)])$, **sense**$(b, [in(b_1, p_3), ..., in(b_4, p_3)])$,**sense**$(a, K_b in(b_1, p_3))$, **sense**$(b, K_a in(b_2, p_1))$.

This sequential plan achieves the goal in spite of the uncertainty of the agents about the world and about the beliefs of the other agents.

Relation to Single Agent Beliefs and DEL

The proposed formulation for handling beliefs in a multiagent setting sits halfway between the standard formulation of beliefs in single agent settings as found in conformant and contingent planning (Geffner and Bonet 2013), and the standard formulation of beliefs in the multiagent settings as found in dynamic epistemic logics (van Ditmarsch, van der Hoek, and Kooi 2007; van Ditmarsch and Kooi 2008). In the single agent settings, beliefs are represented as the sets of states b that are possible, and physical actions a, whether deterministic or not, affect such beliefs deterministically, mapping a belief b into a belief $b_a = \{s \,|\, s \in F(a, s')$ and $s' \in b\}$ where F represent the system dynamics so that $F(a, s)$ stands for the set of states that may follow action a in state s. If the action a is *deterministic*, $F(a, s)$ contains a single state. The belief resulting from doing action a in the belief b and getting an observation token o is $b_a^o = \{s \,|\, s \in b_a$ such that $o \in O(a, s)\}$ where O represents the *sensor model* so that $O(a, s)$ stands for the set of tokens that can be observed after doing action a, resulting in the (possibly hidden) state s. Sensing is noiseless or *deterministic*, when $O(a, s)$ contains a single token. Interestingly, when both the actions and the sensing are *deterministic*, the set of beliefs $B'(t)$ at time t that may follow from an initial belief b_I and a given action sequence can be expressed as

$$B'(t) = \{b(s, t) \mid s \in b_I\} \tag{15}$$

where $b(s, t)$ is the unique belief state that results from the action sequence and the initial belief state b_I when s is the hidden state.

The expression (15) for the set of possible beliefs at time t in the single agent setting, has close similarities with the beliefs $B(t)$ defined by (6) and (7) above

$$B(t) = \{\{v(s, t), r_1(s, t), \ldots, r_m(s, t)\} \mid s \in b_I\} \tag{16}$$

where m is the number of agents, $v(s, t)$ refers to the possible initial state s progressed through the execution up to time t, and $r_i(s, t)$ refers to the set of *possible initial states* that by time t cannot be distinguished from s by agent i. The Kripke structures $\mathcal{K}(t)$ defining the agent (nested) beliefs at time t is built on top of this representation: using such possible initial states s as the possible worlds. It is important to note that it is not possible to define a structure equivalent to $\mathcal{K}(t)$ by identifying the worlds of the structure with the states that are possible at time t. Indeed, the number of belief states that may be possible at time t may be larger than the number of states that are possible at time t, although they can't be larger than the number of states that are possible at time $t = 0$ under the assumptions of *determinism*. *Non-deterministic actions and sensing* can be handled, however, through the usual trick of making such actions and sensors *deterministic* but *conditional* on the value of additional problem variables.

While the proposed formulation is an extension of the belief representation used in single-agent planning, it represents also a *fragment* of *dynamic epistemic logics* where the Kripke structure $\mathcal{K}(t + 1)$ that represents the belief at time $t+1$ is obtained from the Kripke structure $\mathcal{K}(t)$ representing the beliefs at time t and the Kripke structure representing the event at time t called the *event model*. The update operation is known as the *product update* as the set of worlds of the new structure is obtained by taking the cross product of the

sets of worlds of the two time t structures. In particular, using the framework laid out in (van Ditmarsch and Kooi 2008; Bolander and Andersen 2011) for integrating epistemic and physical actions, the *basic actions in our language can be all mapped into simple event models*. The event model for $\mathbf{do}(a)$ is given by a single event whose postcondition in a state s is $f_a(s)$. The event model for $\mathbf{update}(\phi)$ has also a single event with precondition ϕ and null postcondition. Finally, the event model for $\mathbf{sense}(A, \phi)$ has two events that can be distinguished by the agents in A but not by the other agents, one with precondition ϕ, the other with precondition $\neg\phi$, and both with null postconditions. Interestingly, the worlds in Kripke structures that result from this type of events are associated with tuples made of an initial state followed by a sequence of events, and their number does not grow. Thus, belief tracking for this fragment of dynamic epistemic logic remains polynomial in the number of initial states as well. From the perspective of dynamic epistemic logic, hence, our formulation does not add expressive power but rather restricts it without improving the worst-case complexity of this fragment. From a planning perspective, however, the situation is different: it establishes a close connection to the methods used in single-agent planning which in many cases perform much better than what the worst case scenario would suggest. We explore one such connection below, for mapping our *linear multiagent planning problem* into a *classical planning problem* that can be solved by planners off-the-shelf.

Translation into Classical Planning

Let P be a *linear multiagent planning problem*. We show next how to map P into a classical problem $K(P)$ such that the plans for P are plans for its translation $K(P)$, and vice versa, the plans for $K(P)$ are plans for P. The language for $K(P)$ is STRIPS extended with *negation*, *conditional effects*, and *axioms*. This is a PDDL fragment supported by many classical planners. We will use $\neg L$ for a literal L to stand for the complement of L, so that $\neg\neg L$ is L. A conditional effect is an expression of the form $C \rightarrow E$ associated with an action a that states that the head E becomes true when a is applied and C is true. We write such effects as $a : C \rightarrow E$ when convenient. In addition planners normally assume that C and E are sets (conjunctions) of literals. If $C, C' \rightarrow E$ is one such effect, we take $C, \neg C' \rightarrow E$ as a shorthand for the effects $C, \neg L \rightarrow E$ for each literal L in C'. *Axioms* allow the definition of new, derived atoms in terms of primitive ones, called then the primitive fluents. The derived fluents can be used in action preconditions, goals, and in the body of conditional effects. While it's possible to compile axioms away, there are benefits for dealing with them directly in the computation of heuristics and in state progression (Thiébaux, Hoffmann, and Nebel 2005).

For mapping the multiagent problem $P = \langle A, F, I, O, N, U, G\rangle$ into the classical problem $K(P)$, we will make some simplifying assumptions about the types of formulas that may appear in P. We will assume as in planning, and without loss of generality, that such formulas correspond to conjunctions of literals, where a literal L is an (objective) atom p from F or its negation, or an epistemic literal $K_i L$ or $\neg K_i L$ where L is a literal and i is an agent in A. Other formulas, however, can easily be accommodated by adding extra axioms to $K(P)$. We will denote the set of objective literals in P by $L_F(P)$; i.e., $L_F(P) = \{p, \neg p | p \in F\}$, and the set of positive epistemic literals appearing in P by $L_K(P)$; i.e., $L_K(P)$ is the set of $K_i L$ literals that appear as subformula of an action precondition, condition, goal, or sensing or update expression. Indeed, while the set of $K_i L$ literals is infinite, as they can be arbitrarily nested, the set of such literals appearing in P is polynomial in the size of P. As an example, if $\neg K_2 K_1 \neg K_3 p$ is a goal, then $L_K(P)$ will include the (positive epistemic) literals $K_3 p$, $K_1 \neg K_3 p$ and $K_2 K_1 \neg K_3 p$.

The translation $K(P)$ comprises the fluents L/s for the objective literals L in $L_F(P)$, and possible initial states $s \in b_I$, and fluents $D_i(s, s')$ for agents $i \in A$. The former express that the objective literal L is true given that s is the true initial state, while the latter that agent i can distinguish s from s' and vice versa. The epistemic literals $K_i L$ appearing in P, such as $K_3 p$, $K_1 \neg K_3 p$ and $K_2 K_1 \neg K_3 p$ above, are mapped into derived atoms in $K(P)$ through the use of axioms. The expression C/s where C is a conjunction of literals L stands for the conjunction of the literals L/s.

Definition 1. *Let* $P = \langle A, F, I, O, N, U, G\rangle$ *be a* linear multiagent planning problem. *Then the translation* $K(P)$ *of* P *is the* classical planning problem with axioms $K(P) = \langle F', I', O', G', X'\rangle$ *where*

- $F' = \{L/s : L \in L_F(P), s \in b_I\} \cup \{D_i(s, s') : i \in A, s, s' \in b_I\}$,
- $I' = \{L/s : L \in L_F(P), s \in b_I, s \models L\}$,
- $G' = G$,
- $O' = O \cup N \cup U$*; i.e., same set of actions* a *with same preconditions* $Pre(a)$*, but with*
  - *effects* $a : C/s \rightarrow E/s$ *for each* $s \in b_I$*, in place of the effect* $a : C \rightarrow E$ *for physical actions* $\mathbf{do}(a)$, $a \in O$,
  - *effects* $a : C/s, \neg C/s' \rightarrow D_i(s, s'), D_i(s', s)$ *for each pair of states* s, s' *in* b_I *and (parallel) sensing actions* a *in* N *that involve a* $\mathbf{sense}(i, C)$ *expression, and*
  - *effects* $a : \neg C/s' \rightarrow D_i(s, s')$ *and* $a : C/s' \rightarrow D_i(s, s')$*, for each pair of states* s, s' *in* b_I *and* $i \in A$*, for actions* a *of the form* $\mathbf{update}(C)$ *and* $\mathbf{update}(\neg C)$ *respectively,*
- X' *is a set of axioms:*
  - *one for each positive* derived fluent $K_i L/s$ *where* $K_i L \in L_K(P)$ *and* $s \in b_I$ *with (acyclic) definition* $L/s \wedge \wedge_{s' \in b_I}[L/s' \vee D_i(s, s')]$,
  - *one for each literal* L *in* $L_F(P) \cup L_K(P)$ *with definition* $\wedge_{s \in b_I}[L/s \vee D_i(s, s)]$,

In words, the primitive fluents in $K(P)$ represent the truth of the literals L in P conditioned on each possible hidden initial state s as L/s, and the (in)accessibility relation $D_i(s, s')$ among worlds. Initially, the worlds are all accessible from each other and $D_i(s, s')$ is false for all such pairs. On the other hand, L/s is true initially if L is true in s. The goal G' of $K(P)$ is the same as the (conjunctive) goal G of

P, and the actions O' in $K(P)$ are the actions in the sets O, N, and U of P with the same preconditions. However, in the translation, the effect of physical actions is on the L/s literals, while the effect of sensing actions and updates is on the $D_i(s, s')$ literals, with the literals $D_i(s, s)$ for any i being used to denote that the world s is no longer possible. Last, the truth conditions for epistemic literals in the translation is expressed by means of axioms in terms of the primitive literals L/s and $D_i(s, s')$.

The complexity of the translation is quadratic in the number $|b_I|$ of possible initial states. It soundness and completeness properties can be expressed as follows:

Theorem 1. An action sequence π is a plan that solves the linear multiagent planning problem P iff π is a plan that solves the classical planning problem with axioms $K(P)$.

The translation above follows the pattern of other translations developed for conformant and contingent planning problems in the single agent setting (Palacios and Geffner 2009; Albore, Palacios, and Geffner 2009; Brafman and Shani 2012a) and is closest to the one formulated in (Brafman and Shani 2012b). Actually, Brafman, Shani and Zilberstein have recently developed a translation of a class of multiagent contingent planning problems that they refer to as Qualitative Dec-POMDPs (Brafman, Shani, and Zilberstein 2013), as it's a "qualitative" (logical) version of Decentralized POMDPs (Bernstein, Zilberstein, and Immerman 2000). A key difference with our linear multiagent planning problems is that in Q-Dec-POMDPs the agents have beliefs about the world, but not about each other. Hence there are no epistemic modalities or epistemic formulas.

Summary

We have introduced a framework for handling beliefs in the multiagent setting that builds on the methods developed for representing beliefs in single-agent planning. The framework also captures and defines a fragment of dynamic epistemic logics that does not require event models or product updates, and has the same complexity as belief tracking in the single agent setting (exponential in the number of atoms). We have also built on these connections to show how the problem of computing linear multiagent plans can be mapped into a classical planning problem. A basic assumption is that all uncertainty originates in the set of states that are possible initially and hence that actions are deterministic. Still, non-deterministic physical and sensing actions can be introduced by reducing them to deterministic actions whose effects are made conditional on extra hidden variables. In the future, we want to explore the range of multiagent problems that can be effectively solved by existing classical planners, and to develop more compact translations able to exploit width considerations as done in the conformant and contingent settings.

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *Proc. IJCAI-09*, 1623–1628.

Aucher, G., and Bolander, T. 2013. Undecidability in epistemic planning. Technical Report.

Baltag, A., and Moss, L. S. 2004. Logics for epistemic programs. *Synthese* 139(2):165–224.

Baltag, A.; Moss, L. S.; and Solecki, S. 1998. The logic of public announcements, common knowledge, and private suspicions. In *Proc. of the 7th Conf. on Theoretical aspects of rationality and knowledge*, 43–56.

Baral, C.; Gelfond, G.; Pontelli, E.; and Son, T. C. 2012. An action language for reasoning about beliefs in multi-agent domains. In *Proc. of the 14th International Workshop on Non-Monotonic Reasoning*.

Bernstein, D.; Zilberstein, S.; and Immerman, N. 2000. The complexity of decentralized control of Markov decision processes. In *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, 32–37.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proc. IJCAI-01*.

Bolander, T., and Andersen, M. B. 2011. Epistemic planning for single and multi-agent systems. *Journal of Applied Non-Classical Logics* 21(1):9–34.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. of AIPS-2000*, 52–61.

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In *Proc. ICAPS-04*.

Brafman, R. I., and Shani, G. 2012a. A multi-path compilation approach to contingent planning. In *Proc. AAAI*.

Brafman, R. I., and Shani, G. 2012b. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research* 45(1):565–600.

Brafman, R. I.; Shani, G.; and Zilberstein, S. 2013. Qualitative planning under partial observability in multi-agent domains. In *Proc. AAAI*.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning about Knowledge*. MIT Press.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Herzig, A.; Lang, J.; and Marquis, P. 2005. Action progression and revision in multiagent belief structures. In *Proc. 6th Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC 2005)*.

Johnson, M.; Jonker, C.; van Riemsdijk, B.; Feltovich, P.; and Bradshaw, J. M. 2009. Joint activity testbed: Blocks world for teams (bw4t). In *Engineering Societies in the Agents World X*. Springer. 254–256.

Löwe, B.; Pacuit, E.; and Witzel, A. 2011. DEL planning and some tractable cases. In *Logic, Rationality, and Interaction*. Springer. 179–192.

Palacios, H., and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded

Width. *Journal of Artificial Intelligence Research* 35:623–675.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of pddl axioms. *Artif. Intell.* 168(1-2):38–69.

To, S. T.; Pontelli, E.; and Son, T. C. 2011. On the effectiveness of CNF and DNF representations in contingent planning. In *Proc. IJCAI*, 2033–2038.

Van Benthem, J. 2011. *Logical dynamics of information and interaction*. Cambridge University Press.

van Ditmarsch, H., and Kooi, B. 2008. Semantic results for ontic and epistemic change. *Logic and the Foundations of Game and Decision Theory (LOFT 7)* 87–117.

van Ditmarsch, H.; van der Hoek, W.; and Kooi, B. 2007. *Dynamic Epistemic Logic*. Springer.

Multi-Agent Planning with Agent Preferences

Jesús Virseda and **Susana Fernández** and **Daniel Borrajo**
Departamento de Informática
Universidad Carlos III de Madrid
jvirseda@inf.uc3m.es, sfarregu@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract

In the field of Automated Planning, there is a renewed interest in Multi-Agent Planning (MAP). In this paper, we focus on the task of planning efficiently when agents have preferences over goals and want to maintain privacy. Our approach takes as input a MAP task, and a preference of each agent for each goal. Those preferences can be mapped into utilities (rewards) and are modeled as costs (penalties) when planning. The planner has to generate a valid plan taking into account the agent's preferences and the cost of actions. We compare different approaches for assigning goals to agents that heed either the expected costs or preferences, and plan considering either costs or preferences.

Introduction

In the field of Multi-Agent systems, there has been plenty of work on self-interested agents. Most works focus on negotiation, auctions, or single-decision tasks. In Multi-Agent Planning (MAP), planners should also take into account the fact that they have to generate a valid plan (where a sequence of actions is involved, which is different to other sequences of decisions in Multi-Agent systems, as repeated auctions). In MAP, there has not been many approaches yet that focus on self-interested agents. Current work by Nissim and Brafman (2013) focused first on optimal planning, minimizing the cost of plans, to then compute *a posteriori* the payoff (utility) of agents participating in a plan. This payoff is, in other words, a fair payment for each agent proportionally according to its contribution based on plan costs. So, it does not take into account preferences that agents might have *a priori* on achieving specific goals.

In this paper, we focus on a different task, which is based on several past and current projects of our group, where each agent has private information and preferences over goals. We will call it the Multi-Agent Planning with Preferences (MAPP) task. For instance, in logistics transportation domains, each branch of a transportation company has preferences over goals (moving some goods at some place) since they might prefer a transport task over another (García et al. 2013). Likewise, in an ESA (European Space Agency) project, sensors must be assigned (as satellites or antennas) to observe objects based also on some preferences of sensors over objects (Arregui et al. 2012). Finally, in two more projects we dealt with tourist plans where each user has a set of preferences over goals (places to visit) (Cenamor, de la Rosa, and Borrajo 2013; Castillo et al. 2008). So, our main aim is to efficiently find good-quality plans that maximize preference over goals while maintaining agents privacy.

Here, we propose an approach for MAP where agents have explicit preferences over goals, which can be independent of costs to achieve the goals. Solutions ideally would minimize the cost while maximizing the compliance with the preferences of all agents. Here, we extend the distributed approach proposed in (Borrajo 2013b; 2013a) (MAPR) for solving MAP tasks. MAPR first assigns goals to agents and then iteratively solves each agent problem preserving the privacy of agents during the process. Moreover, MAPR flexibility allows us to use any state-of-the-art planner.

In order to reason with preferences, we have to model them into planning tasks. Preferences can be implemented in a positive or negative way. That is to say, if preferences over goals are seen as *rewards* for the agents which achieve the goals, they are positive preferences; and if preferences are seen as *penalties*, they are negative preferences. Since most planners work minimizing total-cost metrics, we use the negative implementation of preferences.

Thus, the problem of MAP with preferences can be understood as a multi-criteria optimization problem, where two different and not related metrics must be minimized: cost and penalty. In our approach, we work with two PDDL domain files and their respective problem files. The *cost* domain and problem model the original planning task, ignoring agent preferences. And, the *penalty* domain and problem ignore action costs. In this latter case, penalties are added in the effects of the actions that achieve goals. We study here the impact in terms of runtime, cost and reward of using both metrics at two different steps in the algorithm: when assigning goals to agents; and during search while the planner is solving each agent MAP task.

The next sections formalize the MAPP task, describe the employed MAPR approach for distributed MAP, give the details of the preferences over goals compilation as penalties, explain the experimental setup and show the results, contrast the differences with related work and finalize with the conclusions of this work.

Multi-Agent Planning Tasks with Preferences

We are interested in MAP tasks with Preferences (MAPP) that can be formally defined as:

Definition 1. *A MAPP task for a set of self-interested agents* $\Phi = \{\phi_i\}_{i=1}^m$ *with private and public information is a tuple* $\Pi = \{F, A, I, G, c, p\}$ *where:*

- $F = F_P \cup \{F_i\}_{i=1}^m$ is a set of fluents that can be public, F_P, or private for each agent, $\{F_i\}_{i=1}^m$
- $A = \{A_i\}_{i=1}^m \cup A_E$ is the set of instantiated actions that agents can perform, $\{A_i\}_{i=1}^m$, and the optional set of *external* actions, A_E, performed by other agents not in Φ
- $I \subseteq F$ is an initial state
- $G \subseteq F$ is a set of goals
- $c : A \rightarrow \Re$ is a cost function representing the cost of every action $a \in A$
- $p : \Phi, G \rightarrow \Re$ is a preference function representing the preference that agent $\phi \in \Phi$ has for achieving goal $g \in G$

Each action $a \in A$ is described by a set of parameters (par(a)), a set of preconditions (pre(a)), that represent literals that must be true in a state to execute the action and a set of effects (eff(a)), literals that are expected to be added (add effects) or removed (delete effects) from the state after the execution of the action. The actions of an agent ϕ_i can be described as: $A_i = \{a \in A \mid \phi_i \in par(a)\}$. And external actions are: $A_E = \{a \in A \mid par(a) \cap \Phi = \emptyset\}$.

The planning task should generate as output a sequence of actions $\pi = (a_1, \ldots, a_k)$ such that if applied in order would result in a state s_k where goals are true: $G \subseteq s_k$. We will later define the plan cost.

A MAPP task Π can be naturally decomposed into a set of partial planning subtasks, one for each agent $\{\Pi_i^p\}_{i=1}^m$ as:

- $\Pi_i^p = \{F_i^p, A_i^p, I_i^p, G_i^p\}$
- $F_i^p = F_P \cup F_i$
- $A_i^p = A_i \cup A_E$
- $I_i^p = I \cap (F_i \cup F_P)$
- $G_i^p = G \cap (F_i \cup F_{P_i})$

where F_{P_i} is the subset of public goals assigned to agent ϕ_i: $\forall i, F_{P_i} \subseteq F_P$ and $\cup_{i=1}^m F_{P_i} = G \cap F_P$. We will see later how we assign a subset of the public goals to each agent. As explained in the next section, our MAP algorithm solves Π by iteratively solving a subset of planning subtasks $M = \{\Pi_1^{p+}, \ldots, \Pi_n^{p+}\}$, $n \leq m$ (since not all agents will need to plan). Each Π_i^{p+} completes Π_i^p with the information communicated by the previous agents in the iteration.

In our work, we consider that privacy is directly related to the information on the state and goals that agents have on a particular domain and problem. Others consider that actions are also public or private, which is not our case. It is only the information on preconditions and effects of actions that is private or public (Nissim and Brafman 2013). Preserving privacy in our work means that agents solve their planning subtasks without ever knowing the private information of other agents. Thus literals $l \in F$ are considered either private or public. In case they are private, they belong to a given agent ϕ_i and they should only be known and modified by ϕ_i when planning. In particular, literals $l \in I$ and $l \in G$ can be private or public in turn. In order to maintain privacy, our planner obfuscates some planning components when agents finish their planning episodes, and communicate them to the following agents, as explained later.

As an example, in the Transport domain of the International Planning Competition (IPC)[1] several vehicles (agents) must transport packages among locations. Fluents derived from the PDDL predicates *(at ?x - vehicle ?v - location)* and *(in ?x - package ?v - vehicle)* and from the function *(capacity ?v -vehicle)* are private. The cost of the action *drive* depends on the road length and the other actions have a cost of 1. In this IPC domain, problems goals only derive from the (at ?x - package ?v - location) predicate. Since the action *drop* is the only one that achieves the *at* predicate, *drop* is the only action achieving goals of Transport problems, and thus the only one that provides rewards to agents. The function p defines the preference every vehicle has for *dropping* every package involved in the goals. There are no external actions in this domain. In other domains, as in the Driverlog where the agents are the drivers, there are external actions, such as *load-truck* and *unload-truck*, since no agent (driver) intervenes in any of the two.

MAPR

MAPR automatically generates the partial planning subtasks $\{\Pi_i^p\}_{i=1}^m$ from the PDDL description of a domain and problem and from the agents description (public goals are assigned to agents at this point). Next, the MAPR algorithm iteratively solves each agent problem. [2] Once an agent solves a problem, it obfuscates the private components of the solution and communicates them to the next agent. In turn, the next agent should solve its own problem augmented with the obfuscated private part of the solution of the previous agents and the public part of those solutions. Therefore, MAPR sees MAP as plan reuse. An important aspect of the algorithm consists on how to assign public goals to agents. As we will explain below, it uses several standard strategies.

Figure 1 shows a high-level description of the algorithm, where we use @ to express obfuscated private information. It takes as input a MAP task (domain, problem and agents description), a goal assignment strategy, the planner to be used by the first agent, and a second planner (it might be the same one) to be used by the following agents. The reason to use two planners (that could be different) is that the second planner might be a replanning system. Since all inputs and outputs are in PDDL, MAPR can use any state-of-the-art planner. The algorithm is then composed of six main steps: goal assignment; first planning episode; obfuscation of the private part of a plan and communicating information to the next agent; merging of a prior agents plan with a planning problem; subsequent planning episodes; and termination. The goal assignment strategy may not assign goals to some of

[1] http://ipc.icaps-conference.org/

[2] For a more detailed description, we refer the reader to (Borrajo 2013b).

the agents and thus these agents are not used in the planning process. As a side comment on the algorithm, in the second and following iterations, when $j = 1$, then $j-1$ means $j = n$ ($n \leq m$). So, the first agent, instead of generating a new plan using the first planner, it takes as input the obfuscated solution from the last agent on the previous iteration. Since MAPR can iterate over each agent once it has completed the first iteration over all agents, MAPR benefits from an implicit backtracking. So, if in the first iteration an agent could not complete its goals due to a wrong decision (such as using a particular action for achieving a goal, or consuming a common resource that another agent needs), MAPR could potentially find a solution if it can be generated by the set of chosen agents.

Function MAPR (M, GA, FP, SP): plan

M: multi-agent planning task
GA: goal assignment strategy
FP: first planner
SP: second planner

Assign subset of public goals to each agent ϕ_i using GA
$\pi_1 \leftarrow$**First-Plan**(FP, Π_1^p)
$j \leftarrow 1$
Repeat until **Termination**
  $j \leftarrow j + 1$
  If $j > n$ Then $j \leftarrow 1$
  ϕ_{j-1} **Obfuscates** its private information, $S^{@}_{j-1}$:
    • the plan $\pi^{@}_{j-1}$ and
    • the problem $\Pi^{@}_{j-1} = \{F^{@}_{j-1}, A^{@}_{j-1}, I^{@}_{j-1}, G^{@}_{j-1}\}$
  ϕ_{j-1} **Communicates** $S^{@}_{j-1}$ to agent ϕ_j
  ϕ_j creates a new planning task, Π_j^{p+}:
    • it **Merges** its assigned problem and $S^{@}_{j-1}$
  $\pi_j \leftarrow$**Second-plan**(SP, Π_j^{p+})
If solved, return last plan

Figure 1: High level description of MAPR planning algorithm.

Goal Assignment

Given the total set of public goals G and a set of agents Φ, MAPR first has to assign a subset of goals to each agent to lower the planning complexity of each individual planning episode. For each goal in $g \in G$ and agent in $\phi_i \in \Phi$, MAPR computes a relaxed plan from the initial state of each agent, I_i, following the well known relaxed plan heuristic of FF (Hoffmann and Nebel 2001). If the relaxed plan heuristic detects a dead-end, then $c(g, \phi_i) = \infty$. This will define a cost matrix, $c(G, \Phi)$. Next, we have devised four goals assignment schemes.

all-achievable: MAPR assigns each goal g to all agents ϕ_i such that $c(g, \phi_i) < \infty$; that is, if the relaxed plan heuristic estimates g could be reached from the initial state of ϕ_i, g is assigned to ϕ_i.

rest-achievable: MAPR assigns goals iteratively. It first assigns to the first agent ϕ_1 all goals that it can reach (cost less than ∞). Then, it removes those goals from the goals set, and assigns to the second agent all goals that it can reach from the remaining set of goals. It continues until the goals set is empty.

best-cost: MAPR assigns each goal g to the agent that can potentially achieve it with the least cost, $\arg\min_{\phi_i \in \Phi} c(g, \phi_i)$

load-balance: MAPR tries to keep a good work balance among agents. It first computes the average number of goals per agent, $k = \frac{|G|}{m}$. Then, it starts assigning goals to agents as in best cost. When it has assigned k goals to an agent, it stops assigning goals to that agent. The next goals that could be assigned to this agent will be redirected to the second best agent for each goal. At the end, agents will have either all k goals, or $m-1$ agents will have k goals and one agent will have the remaining goals, $\mid G \mid -k \times (m-1)$.

In configurations rest-achievable and best-cost, there can be agents for which MAPR does not assign goals.

Obfuscation

If an agent ϕ_j solves its subproblem, then it cannot pass the private information openly to the next agent. So, it obfuscates[3] the private parts and communicates an augmented obfuscated solution $S^{@}_j$ to the next agent. There can be potentially many algorithms for obfuscating the information. In this paper, we use the same simple version of this procedure described in (Borrajo 2013b). Depending on the privacy commitment of the planning task, more complex obfuscating algorithms could be used and the difference will be: more time devoted to the obfuscating algorithm (their time complexity is usually much less than the one of planning); and potentially more space of the obfuscated information (any obfuscating algorithm with a space polynomial complexity could be used without affecting the overall multi-agent planning complexity).

In our case, obfuscating is a two steps process. First, a random substitution is generated for the names of all private predicates, actions and objects. Action names are obfuscated given that, in our privacy preserving scheme, other agents do not need to know the specific actions used by any agent to achieve the goals, even if all information used by those actions is public. As a reminder, in our privacy preserving scheme, actions are not considered public or private; it is only the propositions that are private or public. For instance, in the Satellite domain, if a plan contains an instantiated action as `(calibrate sat1 inst1 Phen6)`, given that `calibrate` and `inst1` are private, MAPR would generate a random substitution as: [4]

$\sigma =$`{(calibrate . g12) (inst1 . g23)}`,
resulting in `(g12 sat1 g23 Phen6)`

The second step in obfuscation consists of applying the substitution to the plan. An augmented obfuscated solution $S^{@}_j$ consists of the obtained plan and the set of components that are needed by the rest of agents to regenerate that solution. More specifically, if the plan of ϕ_j is $\pi_j = (a_1, \ldots, a_t)$, it communicates $S^{@}_j = \{\pi^{@}_j, A^{@}_j, I^{@}_j, G^{@}_j\}$ to ϕ_{j+1}:

[3] We will use obfuscate indistinctly of encrypt.

[4] We are describing the process in the PDDL lifted version, instead in the propositional version.

- the set of instantiated actions in the plan, after obfuscating them, $A_j^{@}$, by obfuscating the actions parameters (par(a_i)), preconditions (pre(a_i)), and effects (eff(a_i)):
  $A_j^{@} = \{a_i^{@} \mid a_i \in \pi_j, a_i^{@} = (\text{par}(a_i)\mid_\sigma, \text{pre}(a_i)\mid_\sigma, \text{eff}(a_i)\mid_\sigma)\}$
  where we use the notation $\alpha\mid_\sigma$ to represent the result of applying substitution σ to formula α.
- the obfuscated plan, $\pi_j^{@} = \{a_1^{@}, \ldots, a_t^{@}\}$, since we can use planning by reuse in the next iteration instead of planning from scratch.
- all goals (private and public, including goals of previous agents), after obfuscating the private ones,[5]

$$G_j^{@} = \{g^{@} \mid g \in G_j, g^{@} = g\mid_\sigma\}$$

- initial state, after obfuscating the private information. Since MAPR only needs to pass to ϕ_{j+1} the relevant private part of the state, it only considers the literals that are preconditions of any action in the plan:

$$I_j^{@} = \{f^{@} \mid f \in I_j, a_i \in \pi_j, f \in \text{pre}(a_i), f^{@} = f\mid_\sigma\}$$

Planning with Preferences

In MAPP tasks, there are two independent metrics: cost and preferences. We deal with cost as in regular planning settings.

Definition 2. *The cost of a plan π is defined as: $C(\pi) = \sum_{a_i \in \pi} c(a_i)$.*

Preferences cannot be handled directly by current planners, as they can only minimize metric values (in fact, most current planners only allow the definition of one metric in a domain file, named `total-cost`). Thus, we have to map preferences to a minimizing criteria, as penalty. Preferences are defined for goals and agents, while metrics are defined in actions. Thus, we have to map preferences into action costs. In order to define the mapping, we first translate agent-goals preferences into actions rewards.

Definition 3. *The reward $r(a_i, g_k, \pi)$ an action a_i receives for achieving a goal g_k in plan π is defined as:*

$$r(a_i, g_k, \pi) = \begin{cases} p(\phi_j, g_k) & \text{if } \phi_j \in par(a_i) \text{ and } a_i \text{ is the last achiever of goal } g_k \text{ in } \pi \\ 0 & \text{otherwise} \end{cases}$$

So, actions only receive a reward if they are the only ones that achieve a top-level goal and there is an agent that executes it. We are assuming that the preference relation is defined for each agent and goal. Now, we can define the total reward that an action receives.

Definition 4. *The total reward an action a_i receives in a plan π is defined as:*

$$r(a_i, \pi) = \sum_{g_k \in G, g_k \in \mathit{eff}(a_i)} r(a_i, g_k, \pi)$$

Now, the reward of a plan is the sum of all rewards obtained by the preferences for goals of the agents that achieved those goals.

[5] Substitution only affects the private goals.

Definition 5. *The total reward $R(\pi)$ of a plan π is defined as $R(\pi) = \sum_{a_i \in \pi} r(a_i, \pi)$.*

Since most planners minimize total-cost metrics, in this paper preferences are converted into penalties (negative preferences) in a standard way.

Definition 6. *The penalty $\rho(a_i, g_k, \pi)$ an action a_i receives for achieving a goal g_k in plan π is defined as:*

$$\rho(a_i, g_k, \pi) = \begin{cases} r_{\max} - r(a_i, g_k, \pi) & \text{if } a_i \text{ is the last achiever of goal } g_k \text{ in } \pi \\ 0 & \text{otherwise} \end{cases}$$

where $r_{\max}$ is the maximum possible reward.

Definition 7. *The total penalty an action a_i receives in a plan π is defined as:*

$$\rho(a_i, \pi) = \sum_{g_k \in G, g_k \in \mathit{eff}(a_i)} \rho(a_i, g_k, \pi)$$

We are interested in agent preferences for achieving goals, so we implement the penalties as increments of the total-cost function only in the actions that achieve goals when they achieve a goal predicate. For example, in the Transport domain where goals derive from the *at* predicate and only the *drop* action has *at* as a positive effect, a new effect *(increase (total-cost) (penalty-vehicle-at ?v ?l ?p))* is added. The init part of the problem contains the values of the *penalty-vehicle-at* function. Only instantiations with the same parameters as some of the goals have values different from 0.

Now, we can define the total penalty of a plan (equivalent to a reward obtained by fulfilling the agents preferences).

Definition 8. *The total penalty $P(\pi)$ of a plan π is defined as $P(\pi) = \sum_{a_i \in \pi} \rho(a_i, \pi)$.*

Note that $r(a_i, g_k, \pi)$ (and consequently $\rho(a_i, g_k, \pi)$ too) includes the condition that a_i is the last one in the plan π that achieves a goal g_k. Thus, this definition is plan dependent. In this paper, we are interested in domains where goals need to be achieved only once and it is not necessary to undo achieved goals. Hence, we can assume that $r(a_i, g_k, \pi) \simeq r(a_i, g_k)$ and $\rho(a_i, g_k, \pi) \simeq \rho(a_i, g_k)$.

Since most state-of-the-art planners only allow the minimization of the *total-cost* function as the unique metric (as, for instance, all planners based on FAST-DOWNWARD (Helmert 2004)), we define two domains for each planning task. As a side note, this is interesting given that the initial idea of defining metrics in PDDL was that domains could reason on different metrics (so all problems in a given domain would use the same domain file) and it would be in the problem where one would define which metric to use for that particular problem. Now, we are forced to define N different domains, one for each metric, while we only need one problem! The first domain is the original one, ignoring agents preferences. And the second domain implements penalties as a total-cost metric. The problem task is common for both domains: the original problem enriched with the penalty information.

Formally, we can say that given a MAPP task $\Pi = \{F, A, I, G, c, p\}$, a MAPP task Π' with *penalty* costs can be obtained by the following transformation:

Definition 9. *Given a MAPP task with action costs and preferences over goals* $\Pi = \{F, A, I, G, c, p\}$ *the equivalent MAPP task with penalty costs can be defined as* $\Pi' = \{F', A, I', G, c'\}$ *with:*

- $F' = F \cup F_p$*, where* $F_p = \{\rho_{ij} | g_i \in G, \phi_j \in \Phi\}$*. Each* ρ_{ij} *is a PDDL function representing the penalty agent* ϕ_j *has for achieving goal* g_i*.*
- $I' = I \cup F_p$
- $G' = G \cup F_p$
- $c' : A \rightarrow \Re^+$ *is a new cost function defined as:*

$$c'(a) = \rho(a, g_i)$$

Therefore, given a MAPP task $\Pi = \{F, A, I, G, c, p\}$, we work with two MAPP tasks: Π' with penalty costs as defined in Definition 9 and $\Pi'' = \{F, A, I, G, c\}$ with standard costs and no preferences. Then, we use the extended MAPR algorithm following four strategies: use Π'' for assigning goals to agents and for planning (cc); use Π' for both (pp); use Π' for assigning goals and Π'' for planning (pc); and use Π'' for assigning goals and Π' for planning (cp). And we can use different metrics to evaluate the quality of the generated plans: plan cost $C(\pi)$, plan reward $R(\pi)$, plan penalty $\rho(\pi)$ and plan utility $U(\pi) = R(\pi) - C(\pi)$ that relates the reward and the cost. In unit-cost domains, plan cost is the length of the solution plan. All measures apply to all strategies. Even if strategies cp and pp ignore action costs when planning, it is possible to compute $C(\pi)$ *a posteriori* by consulting the costs $c(a_i)$ in π. Penalty and reward are opposed values, so it is possible to compute one value based on the other. Also, even if goals are assigned by the agents preferences/costs (so it selects agent ϕ_j because it is the one with highest-preference/lowest-cost over goal g_i) MAPR does not force that in the final plan it is in fact agent ϕ_j that achieves g_i.

This approach is valid only when the goals need to be achieved once. Thus, the actions that are late achievers of goals will be the only achievers of goals (definition of ρ depends on the plan). However, in domains like the Sokoban an agent could place a stone in a goal position, and then another agent might have to move the stone to a different position to fulfill all problem goals. In this case, the reward obtained by the first agent when it places the stone in the temporal goal position should be subtracted from the total reward, so only late achievers get credit. An alternative way to solve a MAPP task Π that avoids this problem is to transform Π into a new task with *soft goals* and *negative utilities* and then compile them away using the technique described in (Keyder and Geffner 2009). Negative utilities stand for conditions to be avoided; for example, a utility $u(p \wedge q)$ = -10 penalizes a plan that results in a state where both p and q are true with an extra cost of 10. We are not using this definition in this paper, since the domains we used in the experiments do not have this problem. However, we provide at least the solution to this problem here. The MAPP task Π_S with soft goals and negative utilities that is equivalent to Π can be obtained by the following transformation:

Definition 10. *Given a MAPP task* $\Pi = \{F, A, I, G, c, p\}$*, the equivalent MAPP task with soft goals and negative utilities is defined as* $\Pi_S = \{F_S, A_S, I, G_S, c, \nu\}$ *where:*

- $G_S = G \cup \{G_p\}_{i=1}^n$*, where* $\{G_p\}_i = \{\gamma_{ij} | g_i \in G, \phi_j \in \Phi\}$ *are the new soft goals* γ_{ij} *representing that agent* ϕ_j *achieves the goal* g_i *with its corresponding preference value* $p(\phi_j, g_i)$
- $F_S = F \cup \{G_p\}_{i=1}^n$
- A_S *transforms every action* $a \in A | (g_i \in$ *eff*$(a)) \wedge (g_i \in G) \wedge (\phi_j \in$ *par*$(a))$ *by adding as a new effect the soft goal* $\gamma_{ij} \in \{G_p\}_i$
- $\nu : \{G_p\}_{i=1}^n \times \{G_p\}_{i=1}^n \rightarrow R^-$ *is the negative utility function defined over every pair* $(\gamma_{ij}, \gamma_{ik})$ *in the following way* $\nu(\gamma_{ij} \wedge \gamma_{ik}) = -p(\phi_j, g_i)$

Properties

The approach we propose to solve preference problems in MAPP inherits the properties of MAPR, i.e. it is suboptimal, sound and incomplete.

Experiments and Results

We have used the following experimental setup for comparison:

Comparing approaches. We compare different distributed strategies against a centralized approach. These distributed strategies are the ones previously defined (cc, pc, cp, and pp). We only show a centralized approach to understand the relation between a distributed approach which preserves privacy and a centralized one that does not preserve privacy. Thus, the centralized approach has an advantage over the distributed approach. The centralized approach is LAMA11, the winner of the last IPC (Helmert 2004; Richter and Westphal 2010). We are interested here on efficient MAPP computation. Therefore, we have used only the first search iteration of LAMA11, that is one run of lazy greedy best first search with actions costs, and FF and LM-cut heuristics with preferred operators. We plan to move into optimal planning or at least improve the quality of the solutions with anytime behavior in the future.

Domains. We have chosen four domains from the previous IPCs that have been regularly used in MAP papers: Rover, Satellite, Transport and Zenotravel. This selection has been done according to our main motivation: domains close to real world problems where agent preferences are relevant. The Transport domain implements action-costs while the other three are unit-cost. The maximum penalty / reward are set as 10 in all of them, because it seems to be intuitive to ask users for preferences in scales from zero to ten. Costs are the original ones used in the IPC and penalties have been generated randomly to ensure independence from the cost values.

Goal assignment. We have used the four defined methods: all-achievable, rest-achievable, load-balance and best-cost.

Planners. We have used the centralized approach explained above for generating the first agent plan and for the successive planning episodes too.

Time and memory bounds. We have used 1800 seconds and 6GB RAM as in the IPC.

Scores. We have used the following metrics, similar to those used in the IPC, to compare the different approaches:

- *Coverage* is the number of solved problems by each approach.
- *Runtime score (S_T)* over a set of problems $\mathcal{P}$, assuming that $T_p^* > 0$, is computed as

$$S_T = \sum_{p \in \mathcal{P}} \frac{1}{1 + \log \frac{T_p}{T_p^*}}$$

  where T_p^* is the minimum time required to solve the problem p by any approach, and T_p is the time required by the approach we want to calculate the score.
- *Cost score (S_C)* over a set of problems $\mathcal{P}$ is computed as

$$S_C = \sum_{p \in \mathcal{P}} \frac{C_p^*}{C_p}$$

  where C_p^* is the minimum cost of any solution of the problem p and C_p is the cost of the solution by the approach we want to calculate the score. This scores is called quality score in the IPC.
- *Reward score (S_R)* over a set of problems $\mathcal{P}$ is calculated as

$$S_R = \sum_{p \in \mathcal{P}} \frac{R_p}{R_p^*}$$

  where R_p^* is the maximum reward achieved in any solution of the problem p and R_p is the reward obtained by the approach we want to calculate the score.
- *(Reward, Cost) pareto-dominance* gives to each approach a score that equals the number of tuples it pareto-dominates for the same problem. (R, C) is said to pareto-dominate (R', C') if and only if $R \geq R'$ and $C \leq C'$. We use this score instead of a utility score to avoid having to normalize the reward and cost values.

We prefer (Reward, Cost) pareto-dominance instead of a utility score (reward - cost), because the utility subtracts two amounts in different metrics. Therefore, the weight of both metrics on the score depends on the domain / problem; whether the plan solutions are long or short, or whether the actions costs are high or not in comparison to the number of goals to achieve, which sets the maximum reward achievable in the problem.

Tables 1, 2, 3 and 4 show the runtime, cost and penalty score results for the domains Rover, Satellite, Transport and Zenotravel, respectively. Table 5 summarizes all the score results in a table.

In terms of coverage, the only domain that presents difficulties is the Transport domain, where only the configurations of the rest-achievable goal selection can solve all problems. However, configurations that use costs when planning with the best-cost goal selection obtain good results in coverage too.

The fastest approach in all domains but Rover uses penalties and the rest-achievable strategy during goal selection and costs for planning. Only the centralized approach using cost metric outperforms it in the Rover domain. Globally, the rest-achievable strategy is the best one when we want to find a solution fast.

Table 1: Results in the Rover domain.

coverage	pc	pp	cp	cc
best-cost	20	20	20	20
load-balance	20	20	20	20
rest-achievable	20	20	20	20
all-achievable	20	20	20	20
centralized		20		20
S_T	pc	pp	cp	cc
best-cost	16.06	15.36	16.74	16.14
load-balance	15.16	14.66	15.01	14.73
rest-achievable	**18.36**	17.31	18.14	17.44
all-achievable	14.93	14.44	14.79	14.53
centralized		18.60		**18.96**
S_C	pc	pp	cp	cc
best-cost	18.00	17.62	18.19	19.18
load-balance	18.73	17.90	17.90	18.73
rest-achievable	18.08	17.29	17.29	18.08
all-achievable	**19.35**	17.83	17.83	**19.35**
centralized		18.41		**19.64**
S_R	pc	pp	cp	cc
best-cost	16.53	19.57	16.62	14.11
load-balance	14.13	17.94	17.94	14.13
rest-achievable	14.64	17.03	17.03	14.64
all-achievable	14.23	**19.77**	**19.77**	14.23
centralized		19.63		14.45
(R, C) pareto-d.	pc	pp	cp	cc
best-cost	85	96	101	76
load-balance	65	95	93	60
rest-achievable	59	69	64	54
all-achievable	62	104	**108**	57
centralized		**149**		87

The best quality plans in all domains, except for the Transport, are obtained by the centralized approach using the cost metric. In the Transport domain, surprisingly, the best configurations are the ones that select goals with the rest-achievable strategy and plan with penalties. The score in the Transport domain by the rest-achievable strategy is clearly influenced by its high coverage. On the other hand, the Transport domain is the only one that implements action-costs (the other three are unit-cost) and the first solution is not the most relevant one to compare the quality of the approaches. Furthermore, the rest-achievable goal selection strategy distributes goals without taking into account the costs / penalties, pruning the search. In the other strategies, the best configurations are the ones that plan using costs, as it was expected. The best-cost goal selection strategy obtains good results in terms of the cost score when it selects the goals using the action-costs and plans with the same metric.

The reward score is the most diverse of all. Globally, the best approach for this metric is the centralized approach using penalties in planning tasks. In the Rover domain, though, the best configurations use the all-achievable goal selection, and plan with penalties. In the Transport domain the best configuration uses the rest-achievable goal selection strategy, and in the Zenotravel domain the best-cost goal selection

Table 2: Results in the Satellite domain.

coverage	pc	pp	cp	cc
best-cost	20	20	20	20
load-balance	20	20	20	20
rest-achievable	20	20	20	20
all-achievable	20	20	20	20
centralized		20		20
S_T	pc	pp	cp	cc
best-cost	15.96	14.16	15.78	16.61
load-balance	15.62	13.87	14.23	15.20
rest-achievable	**19.16**	16.54	17.28	18.24
all-achievable	15.27	13.64	13.89	14.89
centralized		13.81		18.14
S_C	pc	pp	cp	cc
best-cost	17.87	16.67	17.10	17.85
load-balance	17.64	16.72	16.72	17.64
rest-achievable	17.81	16.06	16.06	17.81
all-achievable	**18.01**	15.73	15.73	**18.01**
centralized		16.28		**19.29**
S_R	pc	pp	cp	cc
best-cost	10.76	**17.43**	14.61	9.23
load-balance	10.76	14.99	14.99	10.76
rest-achievable	6.19	12.38	12.38	6.19
all-achievable	9.26	16.55	16.55	9.26
centralized		**17.98**		10.01
(R, C) pareto-d.	pc	pp	cp	cc
best-cost	92	**102**	86	75
load-balance	77	80	81	79
rest-achievable	48	49	45	46
all-achievable	61	57	56	60
centralized		106		**118**

Table 3: Results in the Transport domain.

coverage	pc	pp	cp	cc
best-cost	18	15	17	19
load-balance	16	14	14	17
rest-achievable	**20**	**20**	**20**	**20**
all-achievable	13	12	11	13
centralized		14		17
S_T	pc	pp	cp	cc
best-cost	10.56	7.51	9.79	12.99
load-balance	10.25	7.16	7.19	10.51
rest-achievable	**19.66**	15.90	16.04	18.52
all-achievable	6.35	5.38	4.96	6.37
centralized		5.64		8.42
S_C	pc	pp	cp	cc
best-cost	13.19	10.38	13.11	15.42
load-balance	12.34	10.06	10.06	13.00
rest-achievable	16.16	**17.30**	**17.30**	16.16
all-achievable	9.76	7.52	6.76	9.76
centralized		9.63		13.09
S_R	pc	pp	cp	cc
best-cost	12.38	11.29	12.22	12.92
load-balance	9.74	9.47	9.47	10.47
rest-achievable	**15.04**	15.03	15.03	**15.04**
all-achievable	7.37	8.71	7.79	7.37
centralized		13.98		12.73
(R, C) pareto-d.	pc	pp	cp	cc
best-cost	59	43	75	68
load-balance	30	39	37	42
rest-achievable	122	**128**	**128**	123
all-achievable	12	16	12	15
centralized		61		83

using penalties to select the goals and to plan.

Finally, the (Reward, Cost) pareto-dominance indicates which are the best balanced approaches; those which dominate more often the other ones in both metrics: reward and cost. The best balanced approach is the centralized one, specifically when it uses the costs. In the Transport domain the rest-achievable approaches obtain the best results influenced by the coverage. In the Rover domain, approaches which use penalties to plan are significantly better than those which use costs to plan. On the opposite side is the Zenotravel domain, where the best approaches are which uses costs to plan. In the rest of domains, the results are similar.

Table 5 shows that most MAPP configurations score higher than the centralized approach in runtime. In the other metrics, at least one of the MAPP configurations obtained a score close to the centralized approach. As a reminder, the centralized approach cannot be directly compared against our configurations, given that it does not preserve privacy.

Analyzing the results in more depth, we can affirm that the distributed approaches become to outperforms the centralized ones when the problems come to be more difficult. This fact can be observed in the Transport domain, the most difficult one, where distributed approaches obtain the best results in all the scores. Additionally, approaches that plan using cost get better results in terms of cost score and approaches that plan using penalties get better reward score results. If the goal selection step employs either cost or penalty metric does not seem to affect so much these scores.

Related Work

Most work on multi-agent planning for self-interested agents focuses on finding *stable* solutions in the spirit of game theory (Brafman et al. 2009; Crosby and Rovatsos 2011). Agents have their own goals and are able to form coalitions, costless binding agreements, to fulfill them. A stable solution is a coalition's joint plan such that no subset of its agents would benefit by joining an alternative coalition. Agents' payoffs are computed *a posteriori* and depend on the total cost of the actions carried out by the agents. Nissim and Brafman proposed a privacy-preserving distributed mechanism to find cost optimal solutions that also calculates the payments to each agent (Nissim and Brafman 2013). We model the *self interest* of the agents with the preference function, independently of the cost. And, we calculate suboptimal and not stable solutions.

Oversubscription planning problems also define preferences on the goals (Keyder and Geffner 2009; Smith 2004). They assume it is not possible to achieve all *soft* goals due to

Table 4: Results in the Zenotravel domain.

coverage	pc	pp	cp	cc
best-cost	20	20	20	20
load-balance	20	20	20	20
rest-achievable	20	20	20	20
all-achievable	20	20	20	20
centralized		20		20
S_T	pc	pp	cp	cc
best-cost	15.06	13.73	15.91	16.48
load-balance	14.96	13.80	14.06	14.67
rest-achievable	**19.93**	17.97	18.61	19.11
all-achievable	14.74	13.57	13.76	14.24
centralized		14.19		15.77
S_C	pc	pp	cp	cc
best-cost	16.03	14.60	15.95	17.73
load-balance	16.00	14.54	14.54	16.00
rest-achievable	17.35	15.85	15.85	17.35
all-achievable	**18.28**	14.68	14.68	**18.28**
centralized		14.65		**19.29**
S_R	pc	pp	cp	cc
best-cost	15.25	**18.61**	15.74	14.18
load-balance	14.76	16.21	16.21	14.76
rest-achievable	12.17	11.23	11.23	12.17
all-achievable	15.26	17.80	17.80	15.26
centralized		18.39		15.11
(R, C) pareto-d.	pc	pp	cp	cc
best-cost	98	97	118	118
load-balance	87	83	82	88
rest-achievable	74	77	74	74
all-achievable	**132**	100	100	131
centralized		115		**148**

Table 5: Summary of results in all domains.

coverage	pc	pp	cp	cc
best-cost	78	75	77	79
load-balance	76	74	74	77
rest-achievable	**80**	**80**	**80**	**80**
all-achievable	73	72	71	73
centralized		74		77
S_T	pc	pp	cp	cc
best-cost	57.64	50.75	58.21	62.23
load-balance	55.99	49.50	50.49	55.11
rest-achievable	**77.11**	67.71	70.06	73.31
all-achievable	51.29	47.03	47.40	50.02
centralized		52.24		61.28
S_C	pc	pp	cp	cc
best-cost	65.09	59.27	64.35	**70.20**
load-balance	64.72	59.23	59.23	65.37
rest-achievable	69.41	66.50	66.50	69.41
all-achievable	65.41	55.76	55.00	65.41
centralized		58.96		**71.31**
S_R	pc	pp	cp	cc
best-cost	54.92	**66.90**	59.20	50.45
load-balance	49.39	58.61	58.61	50.12
rest-achievable	48.04	55.67	55.67	48.04
all-achievable	46.12	62.83	61.91	46.12
centralized		**69.98**		52.29
(R, C) pareto-d.	pc	pp	cp	cc
best-cost	334	338	**380**	337
load-balance	259	297	293	269
rest-achievable	303	323	311	297
all-achievable	267	277	276	263
centralized		431		**436**

limited resources. The objective is to find a plan that maximizes the utility, modeled through goal preferences (possibly keeping the cost under a certain bound (García-Olaya, de la Rosa, and Borrajo 2011)). Keyder and Geffner showed that soft goals can be compiled away avoiding the need to devise specific algorithms for handling them (Keyder and Geffner 2009). Unlike oversubscription planning, our work assumes all goals are hard, every one must be achieved. There is some relation, though, given that one could consider that in our case, we could define as soft goals the fact that each agent achieves each goal. But, then, we would also need to specify that all goals are achieved.

Conclusions and Future Work

We have described an approach that deals with the task of Multi-Agent Planning with agents preferences over goals. We describe how to model those preferences as a planning metric to be used by state-of-the-art planners that can only minimize plan costs. Then, the approach divides planning in two main steps: assignment of public goals to agents and planning. Each of these two steps can be configured to take into account either actions costs, or agents preferences modeled as penalties. We show results in several IPC domains with a set of configurations. As expected, results show that approaches that use cost as the main metric when planning are better than those which use penalties when we want to obtain solutions of better quality in terms of cost. The opposite applies when we want to obtain better rewards; then we must employ rewards (translated into penalties) in the planning step. The approaches with more coverage are those which employ the rest-achievable goal selection, and they are the fastest too, because the goals are well distributed and the division of goals does not overload the agents planning iterations. The pareto-dominance depends on the domain structure; in some domains the approaches using costs to plan are better than the others and the opposite applies in other domains.

As future work, we plan to improve the quality of the solutions using an anytime scheme and, later, to move to optimal planning, considering a pareto-optimal search. Also, we want to implement the soft goals compilation defined in (Keyder and Geffner 2009). Using this compilation, the goals of the original problem would remain as hard goals and preferences agents have over goals would be modeled as soft goals.

Acknowledgments

This work has been partially supported by Spanish MICINN project TIN2011-27652-C03-02.

References

Arregui, J. P.; Tejo, J. A.; Linares-López, C.; and Borrajo, D. 2012. Steps towards an operational sensors network planning for space surveillance. In *Proceedings of the SpaceOps'12*.

Borrajo, D. 2013a. Multi-agent planning by plan reuse. Extended abstract. In *Proceedings of the AAMAS'13*, 1141–1142.

Borrajo, D. 2013b. Plan sharing for multi-agent planning. In Nissim, R.; Kovacs, D. L.; and Brafman, R., eds., *Preprints of the ICAPS'13 DMAP Workshop on Distributed and Multi-Agent Planning*, 57–65.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning games. In *Proceedings of IJCAI*, 73–78.

Castillo, L.; Armengol, E.; Onaindía, E.; Sebastiá, L.; González-Boticario, J.; Rodríguez, A.; Fernández, S.; Arias, J. D.; and Borrajo, D. 2008. SAMAP. A user-oriented adaptive system for planning tourist visits. *Expert Systems with Applications* 34(2):1318–1332. ISSN: 0957-4174.

Cenamor, I.; de la Rosa, T.; and Borrajo, D. 2013. Ondroad planner: Building tourist plans using traveling social network information. In *Proceedings of Conference on Human Computation & Crowdsourcing (HCOMP'13). Works-in-Progress & Demonstrations*.

Crosby, M., and Rovatsos, M. 2011. Heuristic multiagent planning with self-interested agents. In *Proceedings of AAMAS*, 1213–1214.

García, J.; Florez, J. E.; Álvaro Torralba; Borrajo, D.; Linares-López, C.; Ángel García-Olaya; and Sáenz, J. 2013. Combining linear programming and automated planning to solve multimodal transportation problems. *European Journal of Operations Research* 227:216–226.

García-Olaya, A.; de la Rosa, T.; and Borrajo, D. 2011. Using relaxed plan heuristic to select goals in oversubscription planning problems. In *Advances in Artificial Intelligence*, volume 7023/2011 of *Lecture Notes on Computer Science*, 183–192. Springer Verlag.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In Shlomo Zilberstein, J. K., and Koenig, S., eds., *Proceedings of ICAPS'04*, 161–170.

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

Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *Journal of Artificial Intelligence Research* 36(1):547–556.

Nissim, R., and Brafman, R. I. 2013. Cost-optimal planning by self-interested agents. In *Proceedings of AAAI'13*.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In *Proceedings of ICAPS'04*, 393–401.

Integrating individual preferences in multi-agent planning

Alejandro Torreño, Eva Onaindia, Óscar Sapena
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia, SPAIN
{atorreno,onaindia,osapena}@dsic.upv.es

Abstract

In this paper we address the problem of incorporating self-interest agents which have individual preferences in a cooperative multi-agent planning (MAP) framework. Thus, agents are aimed at solving together the hard problem goals and, in addition, to satisfy as many as possible of their soft preferences.

We propose an extension of FMAP, an efficient general-purpose forward MAP algorithm that solves cooperative tasks over a diverse set of planning problems, to accommodate the individual preferences of agents. The new heuristic function of FMAP estimates now the cost to reach the common problem goals as well as the utility of a node regarding the individual preferences of the agents. We show some preliminary experimental results of the preference-based FMAP when agents use a Borda voting mechanism to select the best node according to their preference profiles.

Introduction

Multi-Agent Planning (MAP) extends the classical planning paradigm by introducing several independent entities that plan and act together. Over the last years, MAP has primarily focused on studying different types of architectures (distributed or centralized), coordination mechanisms of agents' solutions (coordination before planning, coordination after planning or interleaved planning and coordination) or issues like privay preserving.

Most MAP approaches stem from extensions of the classical single-agent planning paradigm, assuming agents to be fully cooperative. Agents are typically endowed with a set of global goals that must be collectively attained by the group in order to solve the task at hand (Nissim, Brafman, and Domshlak 2010; Torreño, Onaindia, and Sapena 2012; Borrajo 2013).

The inclusion of self-interested agents with individual preferences in MAP is a matter of study rarely addressed by the planning community. Preference-based planning (PBP) is a branch of classical single-agent planning that addresses the problem of determining when a plan is preferred over another (Baier and McIlraith 2009). Users specify the desirable properties of a solution plan in the form of preferences and PBP algorithms search for a solution that satisfies the largest number of preferences.

The combination of PBP and MAP emerges as an interesting and novel field of research. Most of the existing MAP approaches focus only on fully cooperative agents or apply game-theoretic concepts to combine local solutions into a multi-agent solution plan that ensures certain theoretical properties, such as Nash equilibrium (Brafman et al. 2009).

In (Torreño, Onaindia, and Sapena 2013; 2014b), we introduced FMAP (Forward Multi-Agent Planning), a general-purpose MAP framework that efficiently solves cooperative multi-agent planning tasks from different planning domains. FMAP shows to be very effective at solving hard problem instances where the level of interaction between subgoals is strong. FMAP features a refinement planning scheme (Kambhampati 1997), by which agents cooperatively explore a joint search tree. The nodes of the search tree are partial-order plans built through the contributions of one or more planning agents. At each iteration of the procedure, agents pose refinement plans by introducing actions over a base plan chosen among the leaf nodes of the tree through a novel MAP heuristic function. Each agent is provided with an embedded forward-chaining partial-order planner to build refinement plans.

The main limitation of FMAP is that it only attains cooperative tasks in which the goals are known to all the participating agents. This work generalizes the definition of MAP task introduced in (Torreño, Onaindia, and Sapena 2014b), explicitly allowing agents to have individual preferences over the goal state.

In its original form, FMAP agents apply a straightforward A* procedure, selecting the open node of the search tree that minimizes an evaluation function $f = g + h$ as the next base plan to refine. Since FMAP is based on a cooperative scheme, all the agents share the same f value for any given plan.

In this paper, we propose an extension of FMAP to accommodate individual preferences. Our aim is to allow agents to explore the search tree considering not only the global goals, but also their individual preferences. For this purpose, we defined a utility function that allows each agent to estimate the quality of the refinement plans with respect to the global goals as well as its individual preferences.

In single-agent PBP, it is necessary to establish an order-

ing relation that specifies which plans are preferred to others. A possible way to define this relation is to associate a numerical value to each preference, representing the penalty value for the plans that leave such preference unsatisfied. This approach is followed by PDDL3 (Gerevini and Long 2005), in which a preference is violated by a plan if it logically evaluates to false in such plan. As the aforementioned PBP approach, we associate a penalty to each of the preferences and evaluate the quality of the solution plans according to a metric function that takes into consideration the unfulfilled preferences of the agents.

We define a new plan selection scheme that aggregates the individual preferences of the agents when selecting the next base plan to explore. To do so, we rely on concepts from the social choice theory; particularly, we apply voting mechanisms to select the best base plan according to the preference profiles of the agents.

Most of the PBP algorithms follow an incremental approach in which the planner returns a sequence of plans with increasing quality (Baier and McIlraith 2009). For the experimental evaluation, we modified the stop criterion of FMAP, so that agents keep building solution plans of increasing quality after the first one is found.

This paper is organized as follows: next section formalizes a generalized MAP task and presents the main concepts upon which our approach is based, as well as the changes introduced into our specification language to support individual preferences; next, we introduce the modified FMAP algorithm and thoroughly analyze the main changes included; following, we provide some initial experimental results; and finally, we conclude and summarize our upcoming lines of work.

MAP task formalization

Agents in our MAP model work under a limited knowledge of the planning task by assuming that the information not represented in an agent's model is unknown to the agent. The states of the world are modeled through a finite set of *state variables*, $\mathcal{V}$, each of them associated to a finite domain, $\mathcal{D}_v$, of mutually exclusive values that refer to the objects in the world. Assigning a value d to a variable $v \in \mathcal{V}$ generates a *fluent*. A *positive fluent* is a tuple $\langle v, d\rangle$, which indicates that the variable v takes the value d. A *negative fluent* $\langle v, \neg d\rangle$ indicates that v does not take the value d. A *state* is a set of positive and negative fluents.

An *action* is a tuple $\alpha = \langle PRE(\alpha),\ EFF(\alpha)\rangle$, where $PRE(\alpha)$ is a finite set of fluents modeling the preconditions of α, and $EFF(\alpha)$ is a set of *variable assignments* that model the effects of α. Executing an action α in a world state S leads to a new state S' as a result of applying $EFF(\alpha)$ over S.

Definition 1 *A **MAP task** is tuple $\mathcal{T}_{MAP} = \langle \mathcal{AG}, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{A}, \mathcal{P}\rangle$. $\mathcal{AG} = \{1, \ldots, n\}$ is a finite non-empty set of agents. $\mathcal{V} = \bigcup_{i\in\mathcal{AG}} \mathcal{V}^i$, where $\mathcal{V}^i$ is the set of state variables known to an agent i. $\mathcal{I} = \bigcup_{i\in\mathcal{AG}} \mathcal{I}^i$ is a set of fluents that defines the initial state of $\mathcal{T}_{MAP}$. $\mathcal{G}$ is the set of global goals of $\mathcal{T}_{MAP}$ that are common to all the participating agents. $\mathcal{A} = \bigcup_{i\in\mathcal{AG}} \mathcal{A}^i$ is the set of planning actions of the agents. Finally , $\mathcal{P} = \bigcup_{i\in\mathcal{AG}} \mathcal{P}^i$ is the set of preferences of the agents in $\mathcal{T}_{MAP}$.*

Some characteristics of the elements of $\mathcal{T}_{MAP}$ are:

- Since specialized agents are allowed, they may only know a subset of the initial state $\mathcal{I}$. However, the initial states of the agents never contradict each other.
- Tyipically, the sets of actions of two specialized agents are disjoint but they may also contain some common actions.
- $\mathcal{A}$ includes two fictitious actions α_i and α_f: α_i represents the initial state of $\mathcal{T}_{MAP}$, i.e., $PRE(\alpha_i) = \emptyset$ and $EFF(\alpha_i) = \mathcal{I}$, while α_f models the global goals of $\mathcal{T}_{MAP}$, i.e., $PRE(\alpha_f) = \mathcal{G}$, and $EFF(\alpha_f) = \emptyset$.
- The sets of individual preferences of the agents, $\mathcal{P}^i$, are disjoint sets.

The previous definition includes a private set of *preferences* $\mathcal{P}^i$ for each agent i. Preferences in our model are defined as soft goals since they are not required to be accomplished in order to solve the MAP task. Formally, a *preference* of an agent i, $p \in \mathcal{P}^i$, is a tuple $p = \langle f, penalty\rangle$, where f is a fluent that the agent i wants to achieve in $\mathcal{G}$, and $penalty$ is a numerical penalty applied to the agent in case that the preference is not satisfied in a solution plan.

As indicated in Definition 1, our model considers specialized agents such that each agent has a local and limited *view* on the MAP task. The view of an agent includes both the information it knows and the preferences it has on the MAP task at hand.

Definition 2 *The **view** of an agent i on a MAP task is defined as $\mathcal{T}^i_{MAP} = \langle \mathcal{V}^i, \mathcal{A}^i, \mathcal{I}^i, \mathcal{G}, \mathcal{P}^i, \mathcal{T}h^i\rangle$. $\mathcal{V}^i$ is the set of state variables known to agent i; $\mathcal{A}^i \subseteq \mathcal{A}$ is the set of its planning actions; $\mathcal{I}^i$ is the subset of fluents of the initial state $\mathcal{I}$ that are visible to agent i; and $\mathcal{G}$ is the set of global goals of $\mathcal{T}_{MAP}$. All the agents in $\mathcal{T}_{MAP}$ are aware of the global goals of the task. $\mathcal{P}^i$ is the set of individual preferences of agent i, and $\mathcal{T}h^i$ is the acceptable threshold of penalty for the agent to validate a solution plan.*

The state variables of an agent i are determined by the view it has on the initial state, I^i, the planning actions it can perform, A^i, and set of goals of $\mathcal{T}_{MAP}$. This also affects the domain D_v of a variable v. We define $\mathcal{D}^i_v \subseteq \mathcal{D}_v$ as the set of values of the variable v that are known to agent i. Agents in our model interact with each other by sharing information on their state variables. Given a pair of agents i and j, the set of variables they share is defined as $\mathcal{V}^{ij} = \mathcal{V}^{ji} = \mathcal{V}^i \cap \mathcal{V}^j$. Moreover, some of the values in the domain of a variable can also be public to both agents. The set of values of a variable v that are public to a pair of agents i and j is defined as $\mathcal{D}^{ij}_v = \mathcal{D}^{ji}_v = \mathcal{D}^i_v \cap \mathcal{D}^j_v$.

As introduced in (Torreño, Onaindia, and Sapena 2014b), our MAP model is based on a multi-agent refinement planning framework, in which agents apply a Partial-Order Planning (POP) search procedure in order to generate refinement plans. The next definitions briefly introduce standard concepts from the POP paradigm (Ghallab, Nau, and Traverso 2004) adapted to a MAP context with state variables.

Definition 3 *A **partial-order plan** or partial plan is a tuple* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$. $\Delta = \{\alpha | \alpha \in \mathcal{A}\}$ *is the set of actions in* Π. $\mathcal{OR}$ *is a finite set of ordering constraints (*$\prec$*) on* Δ. $\mathcal{CL}$ *is a finite set of causal links of the form* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ *or* $\alpha \overset{\langle v,\neg d \rangle}{\rightarrow} \beta$*, where* α *and* β *are actions in* Δ*. A causal link* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ *enforces a precondition* $\langle v, d \rangle \in PRE(\beta)$ *through an effect* $(v = d) \in EFF(\alpha)$*. Similarly, a causal link* $\alpha \overset{\langle v,\neg d \rangle}{\rightarrow} \beta$ *enforces* $\langle v, \neg d \rangle \in PRE(\beta)$ *through an effect* $(v \neq d) \in EFF(\alpha)$ *or* $(v = d') \in EFF(\alpha)$*,* $d' \neq d$*.*

An *empty* partial plan is defined as $\Pi_0 = \langle \Delta_0, \mathcal{OR}_0, \mathcal{CL}_0 \rangle$, where $\mathcal{OR}_0$ and $\mathcal{CL}_0$ are empty sets, and Δ_0 contains only the fictitious initial action α_i. A partial plan Π for a task $\mathcal{T}_{MAP}$ will always contain α_i.

The introduction of new actions in a partial plan may trigger the appearance of *flaws*, that is, preconditions that are not yet solved through a causal link, and threats. A *threat* over a causal link $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ is caused by an action γ that is not ordered w.r.t. α or β and might potentially modify the value of v ($(v \neq d) \in EFF(\gamma)$ or $(v = d') \in EFF(\gamma)$, $d' \neq d$), making the causal link unsafe.

A *flaw-free* plan is a threat-free partial plan in which the preconditions of all the actions are supported through causal links.

Planning agents in our model cooperate to solve MAP tasks by progressively refining an initially empty plan Π until a solution is found. We define a refinement plan as follows:

Definition 4 *A **refinement plan*** $\Pi_r = \langle \Delta_r, \mathcal{OR}_r, \mathcal{CL}_r \rangle$ *over a partial plan* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$*, is a flaw-free partial plan which extends* Π*, i.e.,* $\Delta \subset \Delta_r$*,* $\mathcal{OR} \subset \mathcal{OR}_r$ *and* $\mathcal{CL} \subset \mathcal{CL}_r$*.* Π_r *introduces a new action* $\alpha \in \Delta_r$ *in* Π*, resulting in* $\Delta_r = \Delta \cup \alpha$*. All the preconditions in* $PRE(\alpha)$ *are linked to existing actions in* Π *through causal links; i.e., all preconditions are supported and so it holds* $\forall p \in PRE(\alpha)$*,* $\exists\, \beta \overset{p}{\rightarrow} \alpha \in \mathcal{CL}_r$*, where* $\beta \in \Delta$*.*

Refinement plans are individually evaluated by each agent to assess not only their quality, but also how they accomplish the agent's preferences. More precisely, an agent i evaluates a refinement plan Π through an evaluation function $f^i(\Pi) = g(\Pi) + selfInterest^i \cdot h_{pub}(\Pi) + (1 - selfInterest^i) \cdot \sum_{\forall p \in \mathcal{P}^i} (\beta_p \cdot h_{pri}(\Pi, p))$, where:

- $g(\Pi)$ is the cost of Π, measured as the number of actions of Π.
- $h_{pub}(\Pi)$ is an estimate of the number of actions required to reach the global goals of the task, $\mathcal{G}$.
- $h_{pri}(\Pi, p)$ estimates the number of actions to satisfy the agent's preference p in the refinement plan Π.
- $\beta_p \in [0, 1]$ is a parameter used to assess the relevance of achieving a particular individual preference p over the others. β_p is defined in terms of the penalty associated to each preference: $\beta_p = penalty(p) / \sum_{\forall p' \in \mathcal{P}^i} penalty(p')$.
- $selfInterest^i \in [0, 1]$ indicates the weight that the agent i gives to the accomplishment of the global goals and its private preferences. The higher the value, the more self-interested the agent, meaning that achieving the individual preferences is more relevant to the agent. In the case of a more cooperative agent that puts the emphasis on achieving the global goals of the task, the value of this parameter will be lower.

For every open node Π (refinement plan) of the search tree, an agent i applies $f^i(\Pi)$, thus creating an individual *preference profile* over the open nodes of the tree. As we will explain in the next sections, agents aggregate the individual preference profiles to select the next base plan to be refined.

Definition 5 *A **solution plan** for* $\mathcal{T}_{MAP}$ *is a refinement plan* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ *that addresses all the global goals* $\mathcal{G}$ *of* $\mathcal{T}_{MAP}$ *and meets the penalty threshold for a majority of the agents. Hence, a refinement plan* Π *is a solution iff* $\alpha_f \in \Pi$ *(and thus, all the goals in* $\mathcal{G}$ *are satisfied; that is,* $\forall g \in PRE(\alpha_f)$*,* $\exists\, \beta \overset{g}{\rightarrow} \alpha_f \in \mathcal{CL}$*,* $\beta \in \Delta$*) and* $|PT| > |\mathcal{AG}|/2$*, where* $PT = \{i \in \mathcal{AG} | PlanPenalty^i(\Pi) \leq \mathcal{T}h^i\}$.

Refinement plans in our model include parallel actions introduced by different agents. As described in (Torreño, Onaindia, and Sapena 2014b), we ensure that the effects and preconditions of such actions are mutually consistent through the resolution of threats over the causal links of the plan. Consistency between any two non-sequential actions introduced by different agents is always guaranteed since refinement plans are flaw-free plans.

Every time an agent i refines a plan by introducing an action $\alpha \in \mathcal{A}^i$, it communicates the resulting refinement plan Π to the rest of the agents in $\mathcal{T}_{MAP}$. As in (Torreño, Onaindia, and Sapena 2014b), privacy is preserved by communicating only the fluents of the new action α that are relevant to the sender and receiver agents. The information of a refinement plan Π that an agent j receives from agent i configures its *view* of such plan, $view^j(\Pi)$.

Extensions to the MAP language

In (Torreño, Onaindia, and Sapena 2014a), we firstly introduced our MAP language based on *PDDL3.1* (Kovacs 2011). We have extended the language with some additional constructs in order to support the extensions introduced in this section. The domain files of the specialized agents do not suffer any modification but the problem file of each agent includes now new constructs to define the behaviour and individual preferences of the agent. Throughout this section, we will illustrate the changes using a simple example from the well-known `driverlog` domain.

Currently, we only allow for the definition of individual preferences regarding the goal state. Therefore, now the `:goal` construct includes the definition of the preferences:

```
(:goal (and
 (= (in package1) s1)
 (= (in package2) s2)
 (preference p0 (= (at driver1) s1))
 (preference p1 (= (pos truck1) s1))
))
```

As shown in the previous example, each preference is associated to an identifier that is then used in the specification of the problem metric and the penalties that are applied to the agent when its preferences are not accomplished. The `:metric` section is defined as in *PDDL2.1* (Fox and Long 2003).

```
(:metric minimize
  (+ (total-time)
     (is-violated p0)
     (* (is-violated p1) 2)
))
```

The above example shows that the metric minimizes the sum of the plan duration and the penalties associated to each preference. Since FMAP does not explicitly manage time, the `total-time` parameter is interpreted as the *makespan* or duration of the plan. In the example, $penalty(\texttt{p0}) = 1$ and $penalty(\texttt{p1}) = 2$.

Finally, we include the `:behaviour` construct to define both the agent's self-interest level and its metric threshold. This section is defined as in the following example:

```
(:behaviour
  (self-interest 0.5)
  (metric-threshold 0)
)
```

The agent in this example gives the same level of relevance to the global goals and its preferences when it evaluates refinement plans. The value of the metric threshold indicates that the agent will only accept a plan as a solution when all its individual preferences are satisfied.

Preference-based FMAP

This section describes the preference-based FMAP algorithm, which was originally designed for fully cooperative agents (Torreño, Onaindia, and Sapena 2014b) and has been revised to accomodate individual preferences. FMAP agents build a joint search tree in which nodes are refinement plans (partial-order plans) whose actions are contributed by one or more planning agents. Each agent independently devises refinement plans over a centralized base plan through an embedded forward-chaining POP (FPOP) procedure.

Algorithm 1 shows the preference-based FMAP algorithm as executed by an agent i. The main stages of the procedure can be summarized as follows:

- **Individual refinement plan generation**: each agent individually applies its embedded FPOP procedure to generate a set of refinement plans over the current base plan, Π_b. In Algorithm 1, the RP^i set stores the refinement plans devised by agent i. A refinement plan introduces a new fully-supported action in Π_b.
- **Communication of refinement plans**: agents communicate each other the refinement plans they generated. Agent i in Algorithm 1 sends each other agent j $view^j(\Pi_i)$, for each $\Pi_i \in RP^i$, thus occluding the information that is private to j. In turn, agent i receives, from each other agent j, $view^i(\Pi_j)$, for all $\Pi_j \in RP^j$. The $Refinements^i$ set stores the view an agent i has on all the refinement plans created by the participating agents in a particular iteration of FMAP.
- **Evaluation of the refinement plans**: each agent i individually applies the utility function f^i, described in the previous section, to evaluate the refinement plans, and it stores them into the $openNodes^i$ set. This set keeps the plans ordered according to the agent's utility function f^i. Agents make use of a DTG-based heuristic function, h_{DTG} (Torreño, Onaindia, and Sapena 2014b), to calculate the heuristic estimates of f^i.
- **Base plan selection**: agents select, among all the open nodes of the multi-agent plan-space search tree, the refinement plan that is preferred by the group of agents. To do so, agents aggregate their individual preferences on the open nodes by means of a social choice mechanism.

Algorithm 1: Preference-based FMAP algorithm as applied by an agent i

$SolutionPlans \leftarrow \emptyset$
$openNodes^i \leftarrow \emptyset$
$\Pi_b \leftarrow \Pi_0$
repeat
  $RP^i \leftarrow FPOP(\Pi_b)$
  $Refinements^i \leftarrow RP^i$
  for $j \in \mathcal{AG}$, $j \neq i$ **do**
    $\forall \Pi^i \in RP^i$, send $view^j(\Pi^i)$ to j
    $\forall \Pi^j \in RP^j$, receive $view^i(\Pi^j)$ from j
    $Refinements^i \leftarrow Refinements^i \cup RP^j$
  $\forall\, \Pi_r \in Refinements^i$, compute $f^i(\Pi_r)$
  $openNodes^i \leftarrow openNodes^i \cup Refinements^i$
  $\Pi_b \leftarrow SocialChoice(openNodes^i)$
  $openNodes^i \leftarrow openNodes^i \setminus \Pi_b$
  if $\alpha_f \in \Pi_b$ **then**
    if $MajorityApproval(\Pi_b)$ **then**
      $SolutionPlans \leftarrow \Pi_b$
until $Timeout \vee openNodes^i = \emptyset$
return $SolutionPlans$

If a base plan Π_b supports the task goals $\mathcal{G}$, i.e., $\alpha_f \in \Pi_b$, agents vote to decide if such a plan is accepted as a solution ($MajorityApproval(\Pi_b)$ function in Algorithm 1). In order for the plan Π_b to be accepted, the sum of the agent's penalties caused by Π_b, $PlanPenalty^i(\Pi_b)$, has to be equal or lower than the agent's threshold $\mathcal{T}h^i$ for a majority of agents. Otherwise, the agents keep searching for plans that are compliant with the threshold of more than half of the agents.

The original, cooperative FMAP algorithm ended its execution after finding a solution plan. For the preference-based version, we modified the stop criterion, allowing agents to proceed searching for better solution plans until a timeout is reached. This feature will be used in the experimental results to better assess the quality of the different solution plans obtained.

In the following, we provide more insight on how an agent i performs the individual evaluation of a refinement plan, and how social choice is applied to select a base plan that is preferred by the group of agent.

Evaluating refinement plans

In the cooperative FMAP version, refinement plans were evaluated by the agent that generated them. At the plan communication stage, all the agents received the result of the evaluation along with the refinement plan. In the preference-based version of FMAP, it is not possible to follow such a scheme, since the utility function introduced requires the refinement plans to be evaluated independently by each of the agents.

As shown in the formalization, given a plan Π, an agent i applies a utility function $f^i(\Pi) = g(\Pi) + selfInterest^i \cdot h_{pub}(\Pi) + (1 - selfInterest^i) \cdot \sum_{\forall p \in \mathcal{P}^i}(\beta_p \cdot h_{pri}(\Pi, p))$ to evaluate how a plan Π adjusts to the global goals and agent i's interests. The $g(\Pi)$ parameter stands for the number of actions of the plan; it is thus common to all the participating agents. The $selfInterest^i$ and β values are straightforwardly infered from the agent's problem file.

The heuristic values, $h_{pub}(\Pi)$ and $h_{pri}(\Pi, p)$, $\forall p \in \mathcal{P}^i$, are jointly estimated by each agent individually. In (Torreño, Onaindia, and Sapena 2014b), we introduced our novel MAP heuristic function, which is based on the concept of Domain Transition Graphs (DTGs) (Helmert 2004). The idea behind this function is to take account of the number of actions of a relaxed plan built in a backwards fashion between the frontier state of the refinement plan, $FS(\Pi)$, and the set of goals of $\mathcal{T}_{MAP}$, $\mathcal{G}$. The frontier state of a plan Π, $FS(\Pi)$, is the set of fluents $\langle v, d \rangle$ achieved by applying the actions in Π over the initial state of $\mathcal{T}_{MAP}$, $\mathcal{I}$.

We have modified the DTG heuristic function to jointly estimate both the number of actions required to reach the global goals, $\mathcal{G}$, and each of the agent's preferences. The procedure first builds the relaxed plan for $\mathcal{G}$ as described in (Torreño, Onaindia, and Sapena 2014b), and stores in $h_{pub}(\Pi)$ the number of actions of the relaxed plan. Next, the preferences of the agents are arranged according to their associated penalties, from the highest to the lowest penalty value.

The heuristic function processes the preferences in order. For each preference $p \in \mathcal{P}^i$, the procedure reuses the existing relaxed plan and adds the necessary actions for p to be reached. Once p is processed, the number of extra actions added to the relaxed plan to support this preference are summed up and returned as $h_{pri}(\Pi, p)$. The procedure uses all the information in the relaxed plan to analyze the next preferences, both the actions required to reach the goals in $\mathcal{G}$ and the ones introduced to support the previously processed preferences.

Selecting a base plan

A key aspect of the preference-based FMAP algorithm is related to the base plan selection stage. At each iteration, the agents select the refinement plan that best accommodates to the global goals and their preferences as the next base plan. Selecting such a plan implies aggregating the individual preference profiles of the agents into a global one.

Social choice theory attains the problem of a set of agents selecting a single outcome among a set of candidates according to their individual preferences (Shoham and Leyton-Brown 2009). More precisely, a *social choice function* selects a single outcome from a set of preference profiles, while a *social welfare function* aggregates a set of preference profiles into a single one. Therefore, the tools and mechanisms provided by the social choice theory fit into the problem of a set of self-interested agents selecting a base plan.

Social choice mechanisms are democratic voting systems by which the alternative preferred by the voters is chosen. One key result in social choice theory is the Arrow impossibility theorem, which states that, given three or more candidates, there is not a social welfare function that meets a specific set of criteria, namely Pareto efficiency and independence of irrelevant alternatives, without being dictatorial (Arrow 1951).

In practice, the previous result shows that there is not an ideal voting method, as it is not possible to simultaneously meet all the desirable social choice theoretical properties. Voting methods can be classified as follows:

- **Simple or sequential election**: in these methods, agents select a single outcome among their preference profiles. These mechanism perform one or more election rounds, so that each agent selects a single candidate in each round and the result of the voting is always a single winner.
- **Ranking methods**: these voting systems allow each agent to provide a complete or partial preference profile. A ranking method aggregates the agents' individual preference profiles into a joint preference profile. The first-classified outcome in the global preference profile is selected as the winner.
- **Condorcet methods**: a subset of the ranking methods accomplish the Condorcet criterion, which can be enunciated as follows: given two outcomes o_1 and o_2, if o_1 is preferred to o_2 by a majority of agents, a Condorcet-compliant method will keep the preference $o_1 \prec o_2$ (Shoham and Leyton-Brown 2009). This property also ensures that the Condorcet winner is also Pareto efficient. On the other hand, finding a Condorcet winner is not always possible, and thus, it is necessary to define a tie-breaking mechanism. Moreover, Arrow theorem shows that Condorcet methods are not independent of irrelevant alternatives (Arrow 1951).
- **Rating methods**: these mechanisms allow agents to provide a rating for each plan in their preference profiles, which makes them more flexible than ranking methods.

Different social choice methods can be tested in FMAP for the agents to jointly select the next base plan. Next section shows the preliminary results obtained after testing the preference-based FMAP along with a ranking strategy.

Experimental results

This section provides the preliminary experimental results we collected. The tests assess the performance of the

Domain	Pfile	First solution plan					Best solution plan				
		Actions	Makespan	Metric	Time	Iter	Actions	Makespan	Metric	Time	Iter
Driverlog	1	13	9	10	3,66	177	20	7	7,5	56,54	3300
	2	13	7	8	4,91	232	19	7	7	173,58	8784
	3	10	7	8	1,20	42	13	7	7	6,38	291
	4	12	5	6,3	1,82	51	13	5	5,7	81,89	2778
	5	13	7	8,3	3,30	81	16	7	7,7	27,94	928
Elevators	1	11	7	8,3	3,26	63	13	7	7,3	19,12	436
	2	6	6	7	7,41	119	10	3	3,3	71	1513
	3	4	4	5,3	0,68	5	8	4	4,7	7,26	118
	4	11	6	7	20,00	355	-	-	-	-	-
	5	17	8	9,3	57,75	990	17	7	8	75,78	1289
Zenotravel	3	6	5	5,5	1,22	27	8	4	4	15,07	433
	4	5	4	5	1,63	41	10	4	4	24,64	845
	5	9	5	6	3,75	103	11	5	5,5	9,67	295
	6	6	3	4	1,26	15	10	3	3,5	12,03	300
	7	14	7	8	14,29	398	16	7	7	101,29	2929

Table 1: Experimental results

preference-based FMAP system by using three different planning domains adapted from the International Planning Competitions[1] (IPC) benchmarks.

FMAP is entirely encoded in Java, and it makes use of *Magentix2*[2] (Such et al. 2012), a middleware multi-agent platform that provides the communication services required by the agents. Magentix2 agents communicate by means of the FIPA Agent Communication Language (O'Brien and Nicol 1998). Messaging is carried out through the Apache QPid broker[3].

All the experimental tests were performed on a single machine with a quad-core Intel Core i7 processor and 8 GB RAM. This machine runs both the complete FMAP system and the QPid broker. We set up the planner by establishing a 5-minute time limit for each planning task, so that agents are allowed to find as many solutions as possible within the time limit. For each task, we take account of the first and best solution (measured by computing the agents' average metric for each solution plan) obtained by the agents within the time limit.

We selected three different domains from the IPC benchmarks and adapted five of the STRIPS tasks to a preference-based MAP context. We modeled some of the goals in the original problems as agent-specific preferences, and added extra preferences so that each agent has two different preferences. All the preferences have 1 unit of associated penalty, and the metric threshold is set to 1, meaning that an agent will approve solution plans that meet at least one of its preferences. The self-interest value is set to 0.5 for all the agents, and thus, they assign the same weight to their preferences and the global goals.

The specific design guidelines we applied to adapt each domain are described as follows:

- **Driverlog** (pfiles 1-5): the `driver` objects of the original domain are defined as agents. The preferences are defined through the predicates `(at-driver driver)` and `(pos truck)`.
- **Elevators** (pfiles 1-5): each `slow-elevator` and `fast-elevator` is defined as an agent. The agents' individual preferences were established using the predicates `(lift-at elevator)` and `(at passenger)`.
- **Zenotravel** (pfiles 3-7): The `aircraft` objects are defined as agents in the *zenotravel* MAP version. Preferences are modeled by means of the predicates `(at aircraft)` and `(in person)`.

Table 1 summarizes the early results we collected. For each MAP task, we provide information about the first and best solution plan found by FMAP according to the average of the agents' metric values. *Actions* and *Makespan* columns refer to the number of actions and duration of the solution plans. *Metric* stands for the average metric value of the solution plan. Finally, *Time* and *Iter* indicate the execution time and number of iterations required by FMAP to find each solution plan.

The social choice method selected for these tests is a Borda voting (Shoham and Leyton-Brown 2009), one of the most common ranking methods. For these experiments, we modified the Borda method in order to use incomplete preference profiles. Using complete preference profiles entails processing all the leaf nodes of the search tree at each iteration of the FMAP procedure, which is too complex even in middle-range tasks. For this reason, we limited the preference profile used by each agent in the voting to a fixed number of 10 candidates.

As shown in Table 1, agents find the first solution that meets the metric thresholds in a matter of seconds (less than one minute in all the tasks). This first solution found by the agents tends to adjust to the metric threshold defined for the majority of agents; that is, a majority of the agents solve one

[1] http://ipc.icaps-conference.org/

[2] http://www.gti-ia.upv.es/sma/tools/magentix2

[3] http://qpid.apache.org/

of their preferences, while the rest of agents do not attain any preference.

The best solution for each task offers a more polished plan that in most cases increases the number of actions, but, in turn, offers an equal or lower duration and attains a higher amount of the agents' preferences. Agents find the best solution in less than a half of the time limit in most cases. In some tasks, the average metric equals the plan duration (makespan), which means that all the preferences of the agents are fulfilled (and thus, the metric only takes account of the plan duration).

Conclusions and future work

In this work, we presented an extension of FMAP, a cooperative MAP system, to support individual preferences. We extended our MAP definition language, based on *PDDL3.1*, to include new constructs to define the agents' preferences and behaviour.

The refinement planning procedure of FMAP has been adapted to a preference-based context. Each agent makes use of a utility function to measure how a plan adjusts to the global goals and its private preferences. The application of this utility function gives rise to an individual preference profile for each of the agents. We apply social theory-based mechanisms in order to aggregate the agents' preference profiles and to select the most appropriate base plan according to the interests of the group of agents at each iteration of the procedure. More precisely, agents apply voting methods to democratically elect the candidate plan that is preferred by the group.

Finally, we provide some early experimental results that focus on solving simple tasks from the IPC benchmarks adapted to a MAP context with preferences. In these experiments, agents apply a Borda voting, a well-known ranking method, to select plans.

This article presents a very early stage of our preference-based MAP work. We have already modified and tested the FMAP framework and the definition language to accommodate individual preferences. However, at this point we have just a single functional coordination method.

We intend to add other more complex social choice schemes that ensure different theoretical properties, such as Condorcet-compliant ranking methods and rating mechanisms. Moreover, we will develop an in-depth experimental comparison to study how the search procedure and the quality of the solution plans are affected by the behavioral parameters defined for each agent, such as the social choice mechanism, the level of self-interest or the metric threshold.

Acknowledgments

This work has been partly supported by the Spanish MICINN under projects Consolider Ingenio 2010 CSD2007-00022 and TIN2011-27652-C03-01, the Valencian Prometeo project II/2013/019, and the FPI-UPV scholarship granted to the first author by the Universitat Politècnica de València.

References

Arrow, K. 1951. Individual values and social choice. *Wiley, New York*.

Baier, J., and McIlraith, S. 2009. Planning with preferences. *AI Magazine* 29(4):25.

Borrajo, D. 2013. Multi-agent planning by plan reuse. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, 1141–1142.

Brafman, R.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 73–78.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. *Technical Report, Department of Electronics for Automation, University of Brescia, Italy*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning. Theory and Practice*. Morgan Kaufmann.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, 161–170.

Kambhampati, S. 1997. Refinement planning as a unifying framework for plan synthesis. *AI Magazine* 18(2):67–97.

Kovacs, D. L. 2011. Complete BNF description of PDDL3.1. Technical report.

Nissim, R.; Brafman, R.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1323–1330.

O'Brien, P., and Nicol, R. 1998. FIPA - towards a standard for software agents. *BT Technology Journal* 16(3):51–59.

Shoham, Y., and Leyton-Brown, K. 2009. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

Such, J.; García-Fornes, A.; Espinosa, A.; and Bellver, J. 2012. Magentix2: A privacy-enhancing agent platform. *Engineering Applications of Artificial Intelligence* 96–109.

Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, volume 242, 762–767. IOS Press.

Torreño, A.; Onaindia, E.; and Sapena, O. 2013. FMAP: a heuristic approach to cooperative multi-agent planning. In *Proceedings of the 1st Workshop on Distributed and Multi-Agent Planning (DMAP 2013)*, 84–92.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014a. A flexible coupling approach to multi-agent planning under incomplete information. *Knowledge and Information Systems* 38(1):141–178.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014b. FMAP: distributed cooperative multi-agent planning. *Applied Intelligence*.

Author Index

Keyword Index