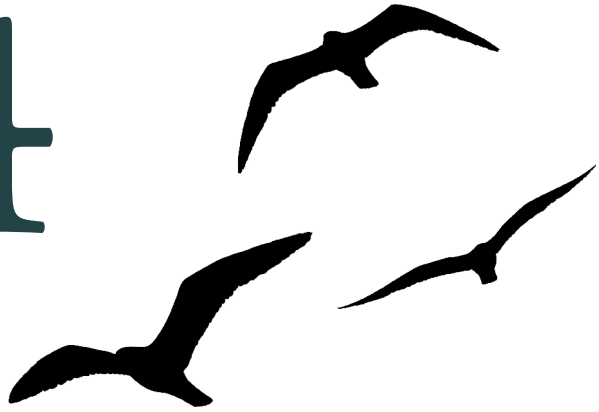
ICAPS 2014



Proceedings of the 8th

Scheduling and Planning Applications Workshop

Edited By:

Gabriella Cortellessa, Mark Giuliano, Riccardo Rasconi and Neil Yorke-Smith

Portsmouth, New Hampshire, USA - June 22, 2014

Organizing Committee

Gabriella Cortellessa
ISTC-CNR, Italy
Mark Giuliano
Space Telescope Science Institute, USA
Riccardo Rasconi
ISTC-CNR, Italy
Neil Yorke-Smith
American University of Beirut, Lebanon, and University of Cambridge, UK

Program Committee

Laura Barbulescu, Carnegie Mellon University, USA
Anthony Barrett, Jet Propulsion Laboratory, USA
Mark Boddy, Adventium, USA
Luis Castillo, University of Granada, Spain
Gabriella Cortellessa, ISTC-CNR, Italy
Riccardo De Benedictis, ISTC-CNR, Italy
Minh Do, SGT Inc., NASA Ames, USA
Simone Fratini, ESA-ESOC, Germany
Mark Giuliano, Space Telescope Science Institute, USA
Christophe Guettier, SAGEM, France
Patrik Haslum, NICTA, Australia
Nicola Policella, ESA-ESOC, Germany
Riccardo Rasconi, ISTC-CNR, Italy
Bernd Schattenberg, University of Ulm, Germany
Tiago Vaquero, University of Toronto, Canada
Ramiro Varela, University of Oviedo, Spain
Gérard Verfaillie, ONERA, France
Neil Yorke-Smith, American University of Beirut, Lebanon and University of Cambridge, UK
Terry Zimmerman, University of Washington – Bothell, USA

Foreword

Application domains that entail planning and scheduling (P&S) problems present a set of compelling challenges to the AI planning and scheduling community, from modeling to technological to institutional issues. New real-world domains and problems are becoming more and more frequently affordable challenges for AI. The international Scheduling and Planning Applications woRKshop (SPARK) was established to foster the practical application of advances made in the AI P&S community. Building on antecedent events, SPARK'14 is the eighth edition of a workshop series designed to provide a stable, long-term forum where researchers and practitioners can discuss the applications of planning and scheduling techniques to real-world problems. The series webpage is at http://decsai.ugr.es/~lcv/SPARK/

We are once more very pleased to continue the tradition of representing more applied aspects of the planning and scheduling community and to perhaps present a pipeline that will enable increased representation of applied papers in the main ICAPS conference.

We thank the Program Committee for their commitment in reviewing. We thank the ICAPS'14 workshop and publication chairs for their support.

The SPARK'14 Organizers

Table of Contents

Optimization Approach for the Management of Transshipment Operations in Maritime Container Terminals

Eduardo Lalla-Ruiz, Christopher Expósito-Izquierdo, Belén Melián-Batista and **J. Marcos Moreno-Vega**
{elalla, cexposit, mbmelian, jmmoreno}@ull.es
Department of Computer Engineering
University of La Laguna, Spain

Abstract

In this paper we propose a functional integration of two well-known logistic problems arising at maritime container terminals. This integration approach is aimed at addressing the berthing operations at the seaside of a container terminal by considering the service time required to serve incoming container vessels in terms of the work plan of the quay cranes. Due to its relevance as practical application in real environments, our interest here is to provide an appropriate framework to optimize the usage of the technical equipment while improving the service of the vessels. The computational results suggest the suitable performance of our scheme in realistic maritime container terminals.

Introduction

A maritime container terminal is an infrastructure aimed at connecting different transportation modes, usually container vessels, trucks, and trains. The main goal of a maritime container terminal is to serve the container vessels arrived to the port. In this regard, serving a container vessel involves to unload those containers that are going to be later retrieved from the terminal by another transportation mode and load those containers to be carried by the vessel towards a different port. The service of a container vessel can be therefore structured into the following operational decisions: (*i*) determining a suitable berthing position and berthing time, (*ii*) allocating an appropriate subset of quay cranes, and (*iii*) scheduling the transshipment operations, that is, loading and unloading of containers (Stahlbock and Voβ 2008).

In this paper we propose an algorithmic integration approach aimed at modelling the service of container vessels arrived to the port. This problem is of utmost importance for terminal managers due to its impact on the overall performance of the terminal.

Application Environment

The berthing operations at a container terminal are aimed at serving the container vessels arrived toward the maritime container terminal. They pursue to maximize the productivity of the handling equipment used to serve the vessels. In this context, two main logistics problems tightly related to each other arise at container terminals:

- *Berth Allocation Problem (BAP)*. This problem is aimed at allocating and scheduling incoming container vessels to berthing positions along the quay of a port (Cordeau et al. 2005).
- *Quay Crane Scheduling Problem (QCSP)*. It is aimed at determining the schedule of the loading and unloading tasks of a container vessel (Bierwirth and Meisel 2009).

The joint consideration of both aforementioned problems might provide a highlighted approach for the management of the seaside operations in maritime container terminals. Therefore, in this work, we address both logistic problems jointly and we will refer to as BAQCSP.

In the BAQCSP, we are given a set of incoming vessels, $V = \{1, ..., v\}$, and a set of berths, $B = \{1, ..., b\}$. Each $j \in B$ is divided into a set of segments, $P_j = \{1, ..., p_j\}$, and has a subset of allocated quay cranes, $Q_j = \{1, ..., q_j\}$. Each $q \in Q_j$ has a position $p \in P_j$ within the berth $j \in B$. Each container vessel $i \in V$ must be assigned to an empty berth $j \in B$ within the vessels and berth time windows, $[tv_i, tv'_i]$ and $[tb_j, tb'_j]$, respectively. The stowage plan defines a set of tasks, $\Omega_i = \{1, ..., n_i\}$ (associated to the loading or unloading operations of the containers groups within the vessel), for each $i \in V$. Each $t \in \Omega_i$ is located in a certain bay along the container vessel, l_t, and has a positive handling time, p_t. It is worth mentioning that it is assumed that each quay crane performs a task without any interruption. This means that once a quay crane starts to (un)load the containers related to a given task, this goes on until all the containers included into the relevant group are (un)loaded. Moreover, in each $j \in B$ each $q \in Q_j$ is only available after its earliest ready time, $r^q \geq 0$, is initially located on a position, l_0^q, and can travel between two adjacent positions of the container vessel with a travel time, $\hat{t} > 0$. The handling time required to serve a vessel $i \in V$ at berth $j \in B$, denoted as c_{ij}, depends on the required time for performing its associated loading/unloading tasks by the quay cranes allocated at that berth. The main goal of the BAQCSP is to determine the berthing position and berthing time of each vessel in order to minimize service time for all vessels, defined as the time elapsed between the arrival of the vessels and the completion of their handling.

Solution Approach

The functional integration approach aimed at providing an overall berth planning for those container vessels arriving to the terminal is as follows:

1. The handling time of each container vessel allocated to a given berth is determined by solving its particular instance of QCSP. Therefore, v times b QCSPs have to be solved in order to obtain the handling times for each vessel within the available berths at the terminal. For solving each QCSP, an Estimation of Distribution Algorithm (EDA) proposed in (Expósito-Izquierdo, Melián-Batista, and Moreno-Vega 2012) is used. The execution of this algorithm is ruled by a maximum computational time, t_{max}. The value of t_{max} is set by the decision maker before starting the execution. $t_{max} = 1$ second in this work.
2. Once the handling times of each vessel are computed, the BAP for determining the berthing position and berthing time of each vessel is addressed. For solving it, a Tabu Search with a Path-Relinking strategy ($T^2S^* + PR$) proposed in (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012) is used.
3. The handling times determined in step 1 may have room for improvement due to the time limit when solving the QCSP. Thus, a limited number of QCSPs restricted to the solution of the BAP obtained in step 2 are considered. Namely, for each vessel constrained to the allocated berth within that solution, a QCSP for that case is solved. Therefore, v QCSPs are solved in this step. The time limit in this case is $\alpha \cdot t_{max}$ seconds, where the value of α is set by the decision maker. $\alpha = 5$ in this work.
4. The solution of the BAP is appropriately adjusted in case the handling times obtained in the previous step have changed. This may be translated into an improvement of the objective function value.

In order to obtain an overall planning and assess the suitability of the proposed integration scheme, a scenario based upon the joint consideration of realistic instances is used. In this regard, the problem instances provided in (Cordeau et al. 2005) and (Bierwirth and Meisel 2009) are tackled.

Figure 1 shows a screenshot of *Seaside Manager* (an experimental software tool that is currently being developed) which illustrates a complete solution for the BAQCSP. In doing so, a representative instance composed of 35 vessels, 10 berths, and 30 quay cranes is considered. At the left vertical axis are delimited the berths and at the right side the number of quay cranes allocated at each berth. The horizontal axis represents the time. Each rectangle corresponds to a container vessel. The specific schedule of the involved loading/unloading tasks is shown for each container vessel.

Concluding Remarks and Future Lines

In this paper, we present a functional integration to jointly address two essential seaside problems at maritime container terminals: the Berth Allocation Problem and the Quay Crane Scheduling Problem. It is noticeable from the computational experiments that the proposed framework provides a feasible

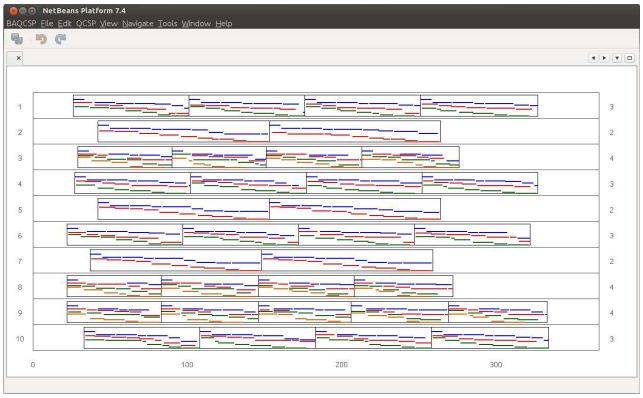


Figure 1: Overall planning of 35 container vessels arrived to a maritime container terminal with 10 berths

overall planning for the seaside operations of a container terminal. In this regard, our approach provides an overall planning for realistic scenarios in less than 40 seconds. Moreover, through the application of efficient approximate algorithms proposed in the related literature, the solution is obtained by means of short computational times. In this regard, the time advantage makes this framework suitable for being included into real decision-support systems.

On the basis of this work, the next stage of our research will be focused on including other realistic objectives functions such as reducing the time of the use of quay cranes, minimizing the idle time of the berths, etc.

Acknowledgments

This work has been partially funded by the European Regional Development Fund, the Spanish Ministry of Economy and Competitiveness (project TIN2012-32608). Eduardo Lalla-Ruiz and Christopher Expósito-Izquierdo thank the Canary Government the financial support they receive through their doctoral grants.

References

Bierwirth, C., and Meisel, F. 2009. A fast heuristic for quay crane scheduling with interference constraints. *Journal of Scheduling* 12(4):345–360.

Cordeau, J.-F.; Laporte, G.; Legato, P.; and Moccia, L. 2005. Models and tabu search heuristics for the berth-allocation problem. *Transportation Science* 39(4):526–538.

Expósito-Izquierdo, C.; Melián-Batista, B.; and Moreno-Vega, M. 2012. Pre-marshalling problem: Heuristic solution method and instances generator. *Expert Systems with Applications* 39(9):8337 – 8349.

Lalla-Ruiz, E.; Melián-Batista, B.; and Moreno-Vega, J. 2012. Artificial intelligence hybrid heuristic based on tabu search for the dynamic berth allocation problem. *Engineering Applications of Artificial Intelligence* 25(6):1132–1141.

Stahlbock, R., and Voβ, S. 2008. Operations research at container terminals: a literature update. *OR Spectrum* 30:1–52.

Temporal Planning for Business Process Optimisation

Daniele Magazzeni
Department of Informatics
King's College London, UK

Fabio Mercorio
CRISP Research Centre
University of Milan Bicocca, Italy

Balbir Barn, Tony Clark, Franco Raimondi
Department of Computer Science
Middlesex University, London, UK

Vinay Kulkarni
Tata Consultancy Services
Pune, India

Abstract

In this paper we consider the problem of designing and optimising a business process. Given a set of activities and a fixed budget, the objective is to determine duration and resource allocation for each activity such that the time-to-market is minimised while budget and dependencies constraints are met. We give a formal description of the problem and we show how it can be cast as a temporal planning problem, resulting in a challenging benchmark planning problem involving concurrency and duration-dependent costs.

The user has to define only dependencies among activities, costs of resources and the available budget, and then use a planner to design an efficient process, which is then generated as a Gantt chart. As a case study, we consider a concrete scenario provided by an industrial partner, and we use a temporal planner to design an effective business process.

Introduction

Business organisations that provide or build products (e.g., software, mixed software-hardware solutions, and even actual goods) are likely to employ abstract modelling languages to describe and analyse their business processes. A number of formal languages are available for modelling business processes, and various tools exist to automate the analysis of the modelled workflows and get advice on how to better invest resources. In a number of instances, including large organisations, the design of business processes is still a manual process that relies on the experience of top-level managers and domain experts to produce sequences of steps that achieve a desired business goal, subject to the minimisation/maximisation of various metrics. As a result, there is no guarantee that the business processes obtained using this process are indeed the most efficient solution. Additionally, the exploration of different options is a very time-consuming task: each new process has to be developed and analysed separately, and alternative solutions need to be compared manually.

In this paper we argue that the design and the optimisation of business processes can be automated using AI planning techniques, thus providing an effective tool to search for "efficient" solutions for resource allocation and task scheduling, or to quickly explore alternative business processes.

More in detail, our contributions can be summarised as follows: we present a novel application domain for temporal planning that is derived form a concrete instance provided by an industrial partner; we describe a benchmark domain that results in a challenging multi-objective temporal planning problem; as a practical example, we consider a model-driven software development process and we show how it can be encoded as a temporal planning problem. Finally, we use a temporal planner to generate solutions that minimise time-to-market for a given project budget and we provide a visual representation of the automatically generated non-trivial resource allocation using Gantt charts.

A Formal Model for Business Processes

In this section we first provide a formal semantics for business processes that abstracts the existing approaches described above. Then, we describe an actual process currently in use at *Tata Consultancy Services*. Then a mapping between the formal model and a temporal planning model is presented, using the concrete business process as a running example.

Definition 1 (Business Process) *A* Business Process (BP) $\mathcal{B}$ *is a 10-tuple* $(\mathcal{S},s_i,s_e,\mathcal{P},\mathcal{D},\mathcal{R},\mathcal{A},\mathcal{T},\mathcal{C},b)$*, where:* $\mathcal{S}$ *is a finite set of* states*,* $s_i \in \mathcal{S}$ *is the* initial state*,* $s_e \in \mathcal{S}$ *is the* end state*,* $\mathcal{P}$ *is a finite set of* parameters*,* $\mathcal{D} : \mathcal{S} \to 2^{\mathcal{S}}$ *is the* dependency function*,* $\mathcal{R} : \mathcal{S} \to 2^{\mathcal{P}}$ *is the* requirements function*,* $\mathcal{A} : \mathcal{S} \to 2^{\mathcal{P}}$ *is the* allocation function*,* $\mathcal{T} : \mathcal{S} \to \mathbb{R}^+$ *is the* temporal function*,* $\mathcal{C} : \mathcal{S} \times 2^{\mathcal{P}} \times \mathbb{R}^+ \to \mathbb{R}^+$ *is the* cost function *and* $b \in \mathbb{R}^+$ *is the* budget*.*

The set of states S corresponds to the set of typical business steps, such as "Requirements analysis" or "Code testing". Each state may depend on other states: for instance, "Code testing" depends on "Code generation". The set $\mathcal{P}$ includes parameters such as number of developers, number of testers, number of domain experts, etc. For each state $s \in \mathcal{S}$, $\mathcal{D}(s)$ defines the set of states s depends on, $\mathcal{R}(s)$ defines the requirements for the phase represented by s, $\mathcal{A}(s)$ defines the resources allocated for it, $\mathcal{T}(s)$ defines its duration and $\mathcal{C}(s, \mathcal{A}(s), \mathcal{T}(s))$ defines its cost. Finally we have an initial state $s_i \in \mathcal{S}$ (such that $\mathcal{D}(s_i) = \emptyset$), and an end state $s_e \in \mathcal{S}$.

Definition 2 (Business Process Execution) *A* Business Process Execution *for the* BP $\mathcal{B}$=$(\mathcal{S},s_i,s_e,\mathcal{P},\mathcal{D},\mathcal{A},\mathcal{T},\mathcal{C},b)$ *is a sequence* $\pi = (s_0a_0t_0)(s_1a_1t_1)(s_2a_2t_2)\dots s_n$*, where,*

$\forall j \geq 0$, $s_j \in \mathcal{S}$, $a_j = \mathcal{A}(s_j)$, $t_j = \mathcal{T}(s_j)$. *A Business Process Execution is* admissible *iff:* (i) $s_0 = s_i$, (ii) $s_n = s_e$, (iii) $\forall j \geq 1, \forall s_k \in \{D(s_j)\} : s_k \in \pi \wedge k < j$, (iv) $\forall s_l \in \mathcal{S}, \mathcal{R}(s_l) \subseteq \mathcal{A}(s_l)$, (v) $\sum_{j=0\ldots n-1} \mathcal{C}(s_j, a_j, t_j) \leq b$.

In particular, conditions (iii), (iv), and (v) require all the dependencies among phases to be satisfied, all phases requirements to be met and the total cost to be within the given budget, respectively.

The Model-Driven Software Development Process

Table 1 describes the states and the associated functions $\mathcal{R}(s), \mathcal{T}(s)$, and $\mathcal{C}(s, a, t)$ for a model-driven software development (MDSD) process that has been used to deliver several large business applications for past 16 years at Tata Consultancy Services.

A row of the table depicts a specific phase of the MDSD process, the time the phase takes to complete as a percentage of total time taken for completion of the MDSD process, and the various actors participating in the phase along with their relative contribution. For instance, the High Level Design phase requires 5% of the time taken by the overall MDSD process, and requires the participation of a Solution Architect (SA), a Domain Expert (DE) and a Technology Architect (TA). If the overall time required by a project is 100 man-days, this line encodes the fact that the HLD phase requires $0.3 \cdot 5$ TA days, $0.1 \cdot 5$ DE days, and $0.6 \cdot 5$ SA days. The remaining actors are Test Engineer (TE), MDE Expert (ME), Modeller (M), Developer (D), and Tester (T).

The initial state is s_i = `RE`, and the end state is s_e = `END`. The dependency function $\mathcal{D}$ is graphically shown in Figure 1. The function $\mathcal{C}(s, a, t)$ can be derived from the times described in Table 1 and the costs in Table 2 (where costs are normalised to the cost of a tester).

Business Process Optimisation as a Temporal Planning Problem

The proposal of this paper is a translation of a business process (as defined in Definition 1) into a temporal planning problem, so that planners can be used to find admissible business process executions (as defined in Definition 2) while minimising the time-to-market.

It is worth noting that the use of *temporal* planning is key in this context as it allows modelling of concurrent activities and time-dependent resource allocations, and to compute duration-dependent costs. Furthermore, although the business process design could be seen as a scheduling problem, it represents an interesting domain where planning plays an important role. To this aim, we follow the same approach proposed in (Fox, Long, and Magazzeni 2011; 2012), where the battery scheduling problem is cast as a temporal planning problem and solved using the temporal planner UPMurphi (Della Penna et al. 2009). In particular, in the battery scheduling problem the number of switching actions cannot be identified in advance as well as in the BP domain the number of resources that can be allocated to each phase is not known in advance. Furthermore, the order in which phases are executed, their duration and how concurrency can be exploited are not know, either. In the following we present the main components of the PDDL domain and problem.

The Planning Domain. The business process domain presents a number of challenging features to be modelled. First, the business process consists of different phases each of which, in turn, requires a number of tasks to be accomplished. The order of execution of the phases is not fixed, but there is a set of dependencies among phases that must be satisfied, which, however, allow for a set of phases to be executed in parallel. Second, each task is associated with a skill and people of different skills have to be allocated to each phase to accomplish their corresponding tasks. The number of people to allocate is not know in advance, and only an upper bound is provided. Third, the project cost, that needs to be maintained within the given budget, depends on how many days each resource is allocated and thus is modelled as a time-dependent cost. Finally, as we want to optimise the time-to-market, the duration of the plan has to be minimised.

We begin the description of the domain with the `start_project` and `end_project` actions, shown in Figure 2a. The start (end) project action is used to enable (disable) the recruitment of resources and the execution of the project phases. Then, in order to model resource allocation, we distinguish between *employing* a resource (which defines the total number of resources for each skill that will be used) and *allocating* a resource (which defines how resources are used throughout the process). We make this distinction as both recruitment and daily costs of resources must be considered.

Employing Resources. Figure 2b shows the `employ` and `dismiss` actions for domain experts (similar actions are defined for other skills). These actions are used for managing the amount of resources recruited over the project. In particular the `employ` action increments the project cost by the cost of recruiting that particular resource (costs of resources of different skills are shown in Table 2) making that available to be allocated. On the other hand, the `dismiss` action, which is applied when the project is completed, is used to dismiss a resource.

Allocating resources. The planner can use allocation (deallocation) actions to assign (release) resources of different skills to each phase of the business process before (after) performing that phase through the corresponding execution action. As an example, Figure 2c shows the actions for allocating and deallocating a domain expert. Note that the `deallocate` action for skill `Y` does not require the whole phase to be finished, but only the task for skill `Y` to be completed. This allows a flexible allocation of the same resource to different phases.

Executing Phases. Modelling the execution of a phase presents an interesting issue, as a phase consists of one or more tasks to be completed. Furthermore, the duration of each task is defined in terms of man-days for the skill required to perform the task (as shown in Table 1). Let us assume that phase `p` requires skills `A`, `B`, `C`, and for each of them the amount of work is `pAdays`, `pBdays` and `pCdays`.

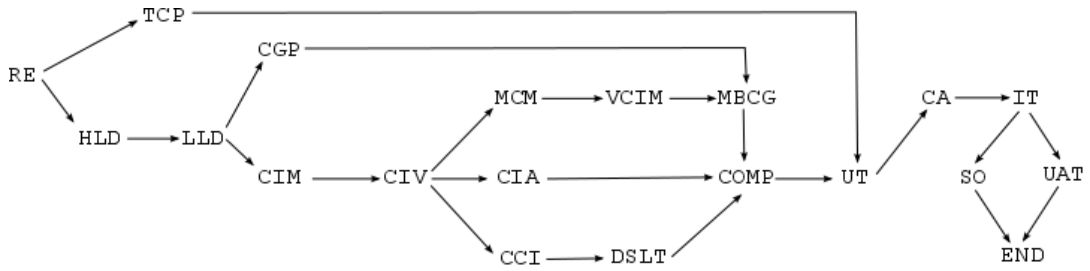


Figure 1: Dependency graph for the MDSD process

Table 1: Phases of the MDSD process

$s \in \mathcal{S}$	$\mathcal{T}(s)$	$\mathcal{R}(s)$							
	% Time	DE	SA	TA	TE	ME	M	D	T
Requirements Elicitation (RE)	15	0.9	0.1						
High Level Design (HLD)	5	0.1	0.6	0.3					
Test Case Preparation (TCP)	5	0.2	0.1		0.7				
Low Level Design (LLD)	10		0.3		0.7				
Code Generator Procurement (CGP)	10		0.2	0.2		0.6			
Component Interface Modelling (CIM)	2		0.1			0.1	0.8		
Component Interface Validation (CIV)	2		0.1			0.1	0.8		
Component Interface Assembly (CIA)	1		0.2			0.1	0.7		
Modelling Component Implementation (MCM)	5		0.1			0.1	0.8		
Validation of Component Implementation Model (VCIM)	5		0.1			0.1	0.8		
Coding of Component Implementation (CCI)	7		0.8					0.2	
Model-Based Code Generation (MBCG)	3		0.1				0.1	0.8	
DSL translation (DSLT)	4		0.1					0.9	
Compilation (COMP)	5							1	
Unit Testing (UT)	5							1	
Component Assembly (CA)	5		0.2				0.4	0.4	
Integration Testing (IT)	5		0.1		0.1				0.8
User Acceptance Testing (UAT)	5	0.1	0.1		0.1				0.7
Sign Off (SO)	1	0.4	0.3		0.3				
END									

```
(:action  start_project
 :parameters (?p - phase)
 :precondition (and
  (todo_project)
  (is_first_phase ?p))
 :effect (and
  (running_project)
  (not (todo_project))))

(:action  end_project
 :parameters (?p - phase)
 :precondition (and
  (completed ?p)
  (is_last_phase ?p))
 :effect (and
  (project_completed)))
```

(a)

```
(:action  employ_DE
:parameters ()
:precondition (and
 (< (employed_DE) (max_DE))
 (running_project))
:effect (and
 (increase (available_DE) 1)
 (increase (employed_DE) 1)
 (increase (total_project_cost)
  (employment_cost_DE))))

(:action  dismiss_DE
 :parameters ()
 :precondition (and (project_completed)
  (> (employed_DE) 0) (> (available_DE) 0))
 :effect (and
  (decrease (employed_DE) 1)
  (decrease (available_DE) 1)))
```

(b)

```
(:action allocate_DE
 :parameters (?p - phase)
 :precondition (and
  (doing ?p)
  (> (available_DE) 0))
 :effect (and
  (increase (allocated_DE ?p) 1)
  (decrease (available_DE) 1)))

(:action deallocate_DE
 :parameters (?p - phase)
 :precondition (and
  (completed_DE ?p)
  (>= (available_DE) 0)
  (> (allocated_DE ?p) 0))
 :effect (and
  (decrease (allocated_DE ?p) 1)
  (increase (available_DE) 1)))
```

(c)

Figure 2: (a) The `start_project` and `end_project` actions. (b) The `employ` and `dismiss` actions for a Design Expert. (c) The `allocate` and `deallocate` actions for a Design Expert

If the planner has allocated `pAres`, `pBres`, `pCres` resources to phase `p`, then the duration of the phase is

$$\max_{i \in \{\mathtt{A},\mathtt{B},\mathtt{C}\}} \left(\frac{\mathtt{p}_i\mathtt{days}}{\mathtt{p}_i\mathtt{res}} \right)$$

Therefore, the effects of the action become effective only when all the tasks have been completed. On the other hand, the resources of skill j allocated for the phase become available as soon as the task requiring skill j terminates, even if the other tasks of the phase are still executing.

Modelling such a scenario is not trivial, and the proposed solution is illustrated in Figure 4. For each phase of the business process, an *envelope* action is used, whose duration is left to the planner, which encapsulates k durative actions (where k is the number of different tasks required to complete the phase), whose duration depends on the resources previously allocated by the planner.

As an example, Figure 3a shows the envelope action to perform a phase, while Figure 3b shows the action for the task requiring domain experts. Note that the number of resources to be allocated to the task is a significant value that the planner will identify carefully. Indeed, this value affects

Table 2: Normalised costs of resources of different skills

Skill	per-day-cost	employment cost
DE	5	30
SA	5	30
TA	4	25
TE	2	20
ME	5	30
M	2	20
D	1.25	12.5
T	1	10

both the duration of a task and, in turn, the cost for executing it, since a resource is paid according to its daily cost and the task duration.

The Planning Problem. The goal is to complete the whole project satisfying all the dependencies. Furthermore, the total project cost must be within the given budget. To this end, the goal has the condition that `total_project_cost` must be no greater than `budget`, where `total_project_cost` depends on the number of resources with different skills allocated to a task and the per-day cost of each resource (as shown in Table 2). As we said before, we are interested in minimising *Time-to-Market*. This is mapped into the planning metric `(:metric minimize (total-time))`.

A fragment of the PDDL problem is shown in Figure 3c, where we show key elements for phase Compilation (COMP) and resource Developer (D). The budget is fixed to $1,500$ (normalised to the cost of a tester). The `per_day_cost_D`, the `employment_cost_D` and the maximum number of Developers that can be employed are defined according to the normalised costs of resources shown in Tab. 2.

The predicate `(todotask_D COMP)` is used to specify which kind of skills are needed to complete the phase (here a Developer is needed to perform the Compilation phase). The dependency graph for the MDSD process is defined through the predicate `(depends COMP CIA MBCG DSLT)` which constrains the execution of the COMP phase to the completion of three distinct phases, that are the Component Interface Assembly, Model-Based Code Generation, and DSL Translation.

Experimental Results

To solve the business process domain, we used the forward chaining temporal planner POPF (Coles et al. 2010). We considered the MDSD of Table 1, with a budget of $1,500$ and a maximum number of employees for each skill to be employed equal to 5. We used a x64 Linux machine equipped with 6 GB of RAM and we considered the best solution found by POPF within 30 minutes[1]. We found a solution for the MDSD process (P in the following) that requires about 15 days, with a total cost of $1,202$.

A complete Gantt chart for the solution P that gives an overview of the execution of phases and tasks has been generated[2]. Here we focus on a fragment of it (shown in Figure 5) which allows us to highlight the key element of the plan, that is the optimised parallel execution of several phases. In particular, the model allows the switching of a resource between phases even when they are still on-going. Specifically, Figure 5 focuses on phases *Modelling Component Implementation* (MCM), *Component Interface Assembly* (CIA), and *Coding of Component Implementation* (CCI). Note that all these three phases need to complete a task which involves Software Architects (boxes with a green vertical texture in Figure 5). Furthermore, the three phases need to be completed in order to start the *Compilation* phase (according to the dependency graph shown in Figure 1). In order to speed up the execution of these parallel phases the planner decides to assign 4 out of 5 Software Architects to complete the task of phase *Coding of Component Implementation*. At the same time, it first assigns the remaining Software Architect to phase *Component Interface Assembly*, and then to phase *Modelling Component Implementation*, while the phase *Coding of Component Implementation* is still running. Through a non-trivial allocation of resources between tasks, the planner is able to reduce the project duration and optimise the budget usage.

Evaluation

The generated solution has been validated by our industrial partner, in comparison with plans used in the company for similar projects. From a practical point of view, our industrial partner confirmed that the added value of the plan-based solution comes from having a plan that suggests efficient durations for phases where resources are switched between ongoing phases, which is key for reducing the time-to-market and that would be hard to be planned manually.

As a further evaluation, we compare the solution P with two other plan-based solutions P_A and P_B, as described in the following.

Solution P_A. First we want to show how planning can provide different high quality solutions according to the amount of resources available, by modifying (if needed) the process itself, and not only the resource allocation. Therefore we modify the planning problem by allowing the planner to recruit up to 6 workers (instead of 5) for each skill. The planner is then able to find the plan P_A, which is more expensive (although still within the assigned budget) but shorter then P. Figure 6 shows a comparison between the two solutions. As can be noticed, the planner produces a different process, and P_A differs from P not only in the resource allocations and phase durations, but also in the order in which the phases are executed.

Solution P_B. Second we want to show how the use of planning can effectively help the business process design, comparing to what one could achieve without using a planner. To this aim, based on the industrial partner's experience, we modify the planning domain to reflect as much as possible how the business process is currently designed in industry. As noticed before, a typical approach followed by man-

[1] POPF is an any-time planner, which improves the current solution as time is given.

[2] Due to the space limitation the complete Gantt chart has been made available at `http://goo.gl/Ki7RvX`

```
(:durative-action execute_phase
:parameters (?p ?dp1 ?dp2 ?dp3 -
phase)
:duration (and (<= ?duration
 (upper_bound ?p)))
:condition (and
   (at start (todo ?p))
   (at start (running_project))
   (at start (depends ?p ?dp1 ?dp2
     ?dp3))
   (at start (completed ?dp1))
   (at start (completed ?dp2))
   (at start (completed ?dp3))
   (at end (completed_DE ?p))
   (at end (completed_SA ?p))
   (at end (completed_TA ?p)))
:effect (and
   (at start (doing ?p))
   (at end (completed ?p))
   (at end (not (doing ?p)))
   (at end (not (todo ?p)))))
```

(a)

```
(:durative-action task-execution_DE
 :parameters (?p - phase
  ?nT - somenumber)
 :duration (= ?duration
  (/ (duration_task_DE ?p)
  (value_of ?nT)))
 :condition (and
  (at start (todotask_DE ?p))
  (over all (doing ?p))
  (over all (= (allocated_DE ?p)
   (value_of ?nT)))
  (at end (>= (employed_DE)
   (value_of ?nT))))
 :effect (and
  (at start (not (todotask_DE ?p)))
  (at end (completed_DE ?p))
  (at end
   (increase
    (total_project_cost)
    (* (* (value_of ?nT) ?duration)
    (per_day_cost_DE))))))
```

(b)

```
(is_first_phase RE) (is_last_phase END)
(= (budget) 1500) (= (total_project_cost) 0)
;; Developer
(= (employed_D) 0) (= (max_D) 5)
(= (available_D) 0) (= (employment_cost_D) 12.5)
(= (per_day_cost_D) 1.25)

;; columns R(s) of Tab. 1 for a Developer
(todotask_D CCI) (todotask_D MBCG)
(todotask_D DSLT)  (todotask_D COMP)
(todotask_D UT)  (todotask_D CA)
(depends COMP CIA MBCG DSLT) ;; graph of Fig.1

;; COMP PHASE
(completed_DE COMP) (completed_TA COMP)
(completed_TE COMP) (completed_ME COMP)
(completed_M COMP) (completed_T COMP)
(completed_SA COMP) (= (duration_task_D COMP) 5)

(:goal (and (project_completed)
  (<= (total_project_cost) (budget)))
(:metric minimize (total-time)))
```

(c)

Figure 3: (a) An example of `execute_phase` action. (b) An example of `task-execution` action. (c) An extraction of PDDL problem for the `COMP` phase

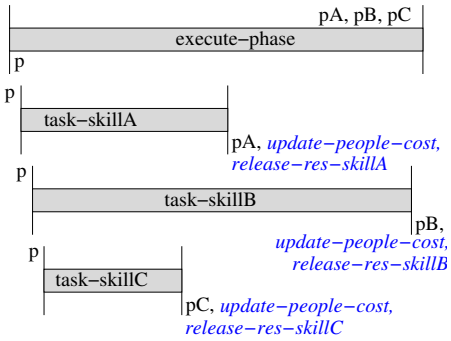


Figure 4: Envelope action for task execution

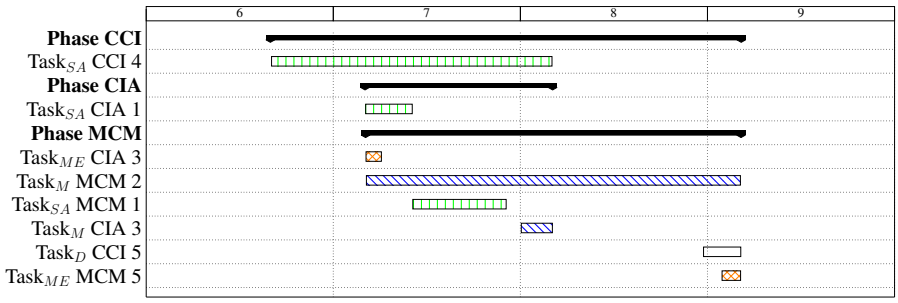


Figure 5: A fragment of the Gannt Chart of Solution P . Relevant tasks are highlighted with a green vertical texture, an orange crosshatched texture or a blue slanted texture, to refer to Software Architects (SA), MDE Experts (ME) or Modellers (M), respectively.

agers is to move a resource to a different phase only when the current phase is finished, as the switching of resources between ongoing phases is hard to be planned manually. To model such a scenario, we modify the `deallocate (?p - phase)` action requiring the whole phase to be finished to release a resource. The best solution found by the planner is plan P_B shown in Figure 7 in comparison with P. As expected, each phase duration is much longer than in plan P, and the whole project takes 41 days longer than in P, when faced with the same budget.

Note that dependencies between phases do not limit the planner in deciding the order in which phases are executed. Indeed, both P_A and P_B solutions present a different execution flow, while continuing to satisfy the dependency graph[3].

Discussion and Related Work

A number of P&S techniques as well as CP-based approaches have been applied to the domain of BPM, aiming to help workflow designers. In particular, FlowOpt (Barták et al. 2011; 2012) is a tool for workflow optimisation, based on constraint satisfaction techniques. The user can visually model a workflow, and the tool automatically generates a production schedule represented as a Gantt chart. The user can then vary the quantities of items to be produced and the tool will modify the schedule accordingly also suggesting some improvements such as buying new resources. On the other hand, JABBAH (Gonzalez-Ferrer, Fernandez-Olivares, and Castillo 2013) provides a mapping between BPM and the HTN planning paradigm. The tool takes as input a workflow graph (described in the BPMN notation) and translates it into HTN-PDDL code. Then the planner IACTIVE is used to find a valid plan, i.e., a valid Gantt chart for the given BPM. In the work by Senkul and Toroslu (Senkul and Toroslu 2005), workflows described in the WSL language are translated into constraint programs in Oz, and then CP techniques are used to find valid resource allocations. Other related approaches include (Lombardi and Milano 2009; Valls, Pérez, and Quintanilla 2009; Wang et al. 2011; Wang and Smith 2005).

Although these papers are all relevant, a key issue that differentiates our work is the *temporal optimisation*. In par-

[3]The complete set of PDDL domain/problems/plans have been made publicly available at `http://goo.gl/OlUDWS`

ticular, we consider tasks with flexible durations whose values are optimised by the planner. Furthermore, the order of tasks is not fixed, but is chosen by the planner, too. This motivates the exploration of using a planner to minimise time-to-market while respecting budget constraints. Furthermore, users have to define only dependencies among phases, leaving to the planner the decision on how to better schedule the activities for improving efficiency while respecting the dependencies.

BPM has also been modelled as a resource-constrained project scheduling problem (RCPSP), where a set of tasks (activities) with fixed start times and durations, have to be executed without interruption using a given set of resources. But again, existing works consider activities with fixed durations (Hartmann and Briskorn 2010). Recently the multi-mode RCPSP framework has been proposed, where an activity can be associated with several modes, each modelling a feasible pair (resource allocation/duration) for the activity. This approach, however, requires the set of possible modes to be enumerated by the user, which represents a significant issue for BPM designers when faced with large projects. Instead, in the planning based approach, this issue is left to the planner which can look for efficient solutions going beyond the limited set of possibilities provided by the user.

Finally, it is worth mentioning the work (Hoffmann, Weber, and Kraft 2012) developed in collaboration with SAP where the planner FF is used for process composition.

Conclusion and Future Work

In this paper we described how the problem of designing and optimising a business process can be cast as a temporal planning problem. Our experience at Tata Consultancy Services over nearly two decades has shown that the correct design of business processes can make the difference between successful and unsuccessful projects. However, in spite of the large body of work available for the generation and verification of business processes (see for instance (Bianculli, Ghezzi, and Spoletini 2007) and references therein), the support for the automatic *optimisation* of business processes is still at an early stage. We presented a contribution on this direction, modelling the design and the optimisation of the business process as a temporal planning domain. We then provided an effective solution for an industrial case study using a temporal planner to find plans that minimise time-to-market for a given project budget.

The domain, as such, represents a challenging problem for planning as it requires concurrency and the handling of durative actions with duration-dependent costs. Beyond that, we hope that the problem we consider in this paper, where activities to be scheduled have a flexible duration dependent on resource allocation and time-to-market needs to be minimised, can represent an interesting benchmark for the community and foster the application of P&S and CP-based techniques to the domain of BPM optimisation.

A natural future work for extending the proposed model is to exploit the expressive power of PDDL3 (Gerevini and Long 2006) and use *preferences* to take into account soft constraints. Furthermore, it will be challenging to deal with the multi-objective optimisation involved in this problem (such as time-to-market vs budget tradeoff) and provide richer suggestions to business organisations. Finally, we want to explore the use of mixed approach where planning and scheduling techniques can be interleaved to find efficient solutions.

References

Barták, R.; Jaska, M.; Novák, L.; Rovensky, V.; Skalicky, T.; Cully, M.; Sheahan, C.; and Thanh-Tung, D. 2011. Workflow optimization with FlowOpt: On modelling, optimizing, visualizing, and analysing production workflows. In *Proc. TAAI'11*, 167–172.

Barták, R.; Jaska, M.; Novák, L.; Rovensky, V.; Skalicky, T.; Cully, M.; Sheahan, C.; and Thanh-Tung, D. 2012. FlowOpt: Bridging the gap between optimization technology and manufacturing planners. In *Proc. ECAI'12*, 1003–1004.

Bianculli, D.; Ghezzi, C.; and Spoletini, P. 2007. A model checking approach to verify BPEL4WS workflows. In *Proc. SOCA'07*, 13–20.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proc. ICAPS*.

Della Penna, G.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2009. UPMurphi: a tool for universal planning on PDDL+ problems. In *Proc. ICAPS'09*, 106–113. AAAI Press.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic construction of efficient multiple battery usage policies. In *Proc. ICAPS'11*, 74–81.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based policies for efficient multiple battery load management. *J. Artif. Intell. Res. (JAIR)* 44:335–382.

Gerevini, A., and Long, D. 2006. Preferences and soft constraints in PDDL3. In *Proceedings of ICAPS Workshop on Planning with Preferences and Soft Constraints*.

Gonzalez-Ferrer, A.; Fernandez-Olivares, J.; and Castillo, L. 2013. From business process models to hierarchical task network planning domains. *The Knowledge Engineering Review* 28(2):175–193.

Hartmann, S., and Briskorn, D. 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 207(1):1–14.

Hoffmann, J.; Weber, I.; and Kraft, F. 2012. SAP speaks PDDL: Exploiting a software-engineering model for planning in business process management. *J. Artif. Intell. Res. (JAIR)* 44:587–632.

Lombardi, M., and Milano, M. 2009. A precedence constraint posting approach for the rcpsp with time lags and variable durations. In *Principles and Practice of Constraint Programming-CP 2009*. Springer. 569–583.

Senkul, P., and Toroslu, I. H. 2005. An architecture for workflow scheduling under resource allocation constraints. *Information Systems* 30(5):399–422.

Valls, V.; Pérez, Á.; and Quintanilla, S. 2009. Skilled workforce scheduling in service centres. *European Journal of Operational Research* 193(3):791–804.

Wang, X., and Smith, S. F. 2005. Retaining flexibility to maximize quality when the scheduler has the right to decide activity durations. In *ICAPS*, 212–221.

Wang, X.; Policella, N.; Smith, S. F.; and Oddi, A. 2011. Constraint-based methods for scheduling discretionary services. *Ai Communications* 24(1):51–73.

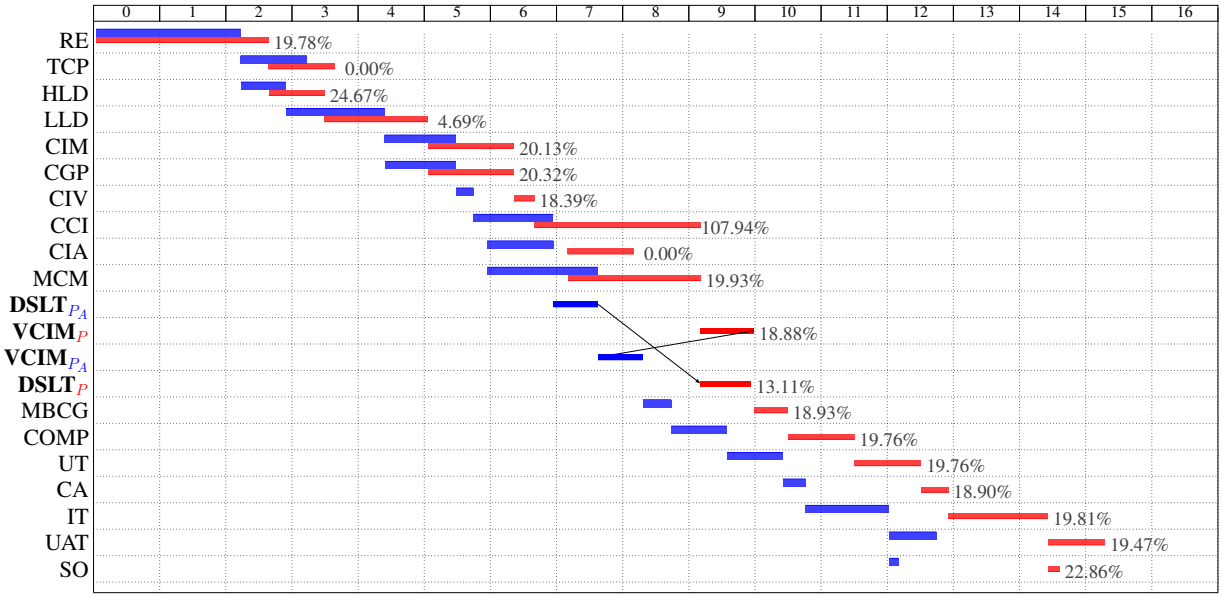


Figure 6: Solution P_A. Plan P (red) using at most 5 workers costs 1,202.405 with time-to-market 15.289. Plan P_A (blue) using at most 6 workers costs 1,338.672 with time-to-market 12.755. The duration of each phase for solution P_A is compared with the corresponding phase of solution P. The difference is shown in percentage with respect to P.

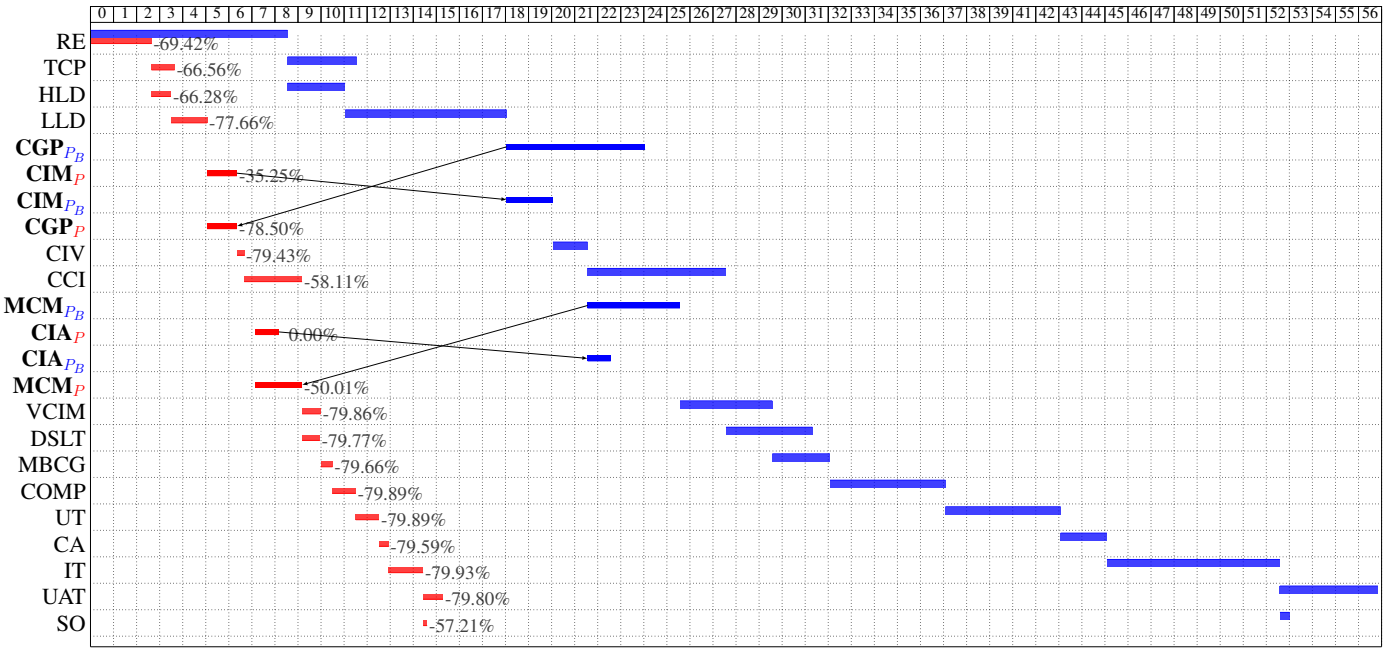


Figure 7: Solution P_B. Plan P (red) using at most 5 workers costs 1,202.405 with time-to-market 15.289. Plan P_B (blue) with no resource exchange between phases costs 1,205.979 with time-to-market 55.851. The duration of each phase for solution P_B is compared with the corresponding phase of solution P. The difference is shown in percentage with respect to P.

New Algorithms for The Top-K Planning Problem

Anton V. Riabov Shirin Sohrabi Octavian Udrea
IBM T.J. Watson Research Center
PO Box 704, Yorktown Heights, NY 10598, USA
{riabov, ssohrab, oudrea}@us.ibm.com

Abstract

Cost-optimal planning is a variant of a general planning problem, where all actions have non-negative costs, and the solution is a valid plan that minimizes the sum of the costs of all actions included in the plan. In this paper, we propose a new planning problem formulation, top-k planning, which is a generalization of cost-optimal planning with applications in plan recognition, diagnosis, explanation generation, and other domains. No existing planners can solve this problem out of the box. We have implemented and compared a total of four new planning algorithms for top-k planning. Two of the algorithms are based on the k shortest paths algorithm by Eppstein and a recently proposed variant of that algorithm for dynamic graphs called K^*, by Aljazzar and Leue. We also implemented a branch and bound algorithm, and an iterative replanning algorithm based on LAMA. Our experiments show that the top-k planning problem can be solved efficiently, in time comparable to cost-optimal planning. We also show that our implementation of top-k planning based on the K^* algorithm outperforms other algorithms.

Introduction

The shortest path problem is a problem of finding a path connecting a given source-destination pair in a graph with minimum total cost (or length). The all-pair shortest path requires computation of the shortest path between all pairs in the graph. Several researches have examined modeling the planning problem as a shortest path computation in a graph or more specifically applying the single-source or all-pair shortest paths algorithm to precompute all shortest paths (e.g., (Botea and Harabor 2013; Edelkamp and Kissmann 2009)). This eliminates the need to search and can lead to fast computation of cost-optimal plans. In turn, cost-optimal planning can also be used for solving a class of preference-based planning problems (Baier and McIlraith 2008) following Keyder and Geffner 2009.

Given an arbitrary number k, the k shortest paths problem is a problem of finding the k shortest paths from a source node to a destination node in a graph. This problem has many applications including path planning (e.g., (Zhu et al. 2013)), video games (Botea 2011), and networking. There are a number of reasons why a k shortest paths algorithm could be needed instead of a single shortest path. Computing k shortest paths can be beneficial, for example, if there are other types of constraints beyond path costs, but those constraints are not fully defined or are missing. In addition, analyzing the k shortest paths can help gain better understanding of the properties of the problem and its optimal solutions. See (Eppstein 1998) for a more comprehensive discussion of these applications.

In this paper, similar to how the k shortest paths extends the shortest path problem, we propose the formulation of the top-k planning problem for cost-optimal planning. We also propose four planning algorithms for top-k planning based on existing methods, including k shortest paths computation for plan cost minimization over the state graph. Unlike optimal planning, where the objective is to find one optimal plan with minimum cost, we define the top-k planning problem as the problem of finding a set of k distinct plans with lowest cost. This set can contain both optimal plans and near-optimal plans, depending on k, and, by definition, for each plan in this set all valid plans of lower cost must also be in the set. To the best of our knowledge we are the first to formulate the top-k planning problem and propose a solution to finding top-k plans, at least for cost-optimal planning.

We argue that the top-k planning problem has important applications, some of which intersect with those of the k shortest paths problem. In particular, we are interested in this problem because of its applications in plan recognition, diagnosis of discrete event systems, and explanation generation, all of which can be modeled as planning problems (e.g., (Ramírez and Geffner 2009; Sohrabi, Baier, and McIlraith 2011)). In these applications it may be important to not only generate one optimal solution, but a set of "good" alternatives. In recent work, the need for top-k planning has been highlighted in the malware detection problem, where the objective is to explain the sequence of observations given the system description (Sohrabi, Udrea, and Riabov 2013). There, top-k plans correspond to alternative plausible hypotheses explaining unreliable observations. Generally, computing top-k plans can help deal with incompleteness in the domain, imperfect quality measures, and unreliable knowledge, such as missing or noisy observations. Hence, we believe top-k planning formulations and algorithms can provide some of the tools needed for fulfilling the model-lite planning vision (Kambhampati 2007).

The k shortest paths problem was introduced in (Hoffman and Pavley 1959) and several efficient algorithms were developed for it. In particular Yen's algorithm (Yen 1971) and

several later implementation improvements of it are used to find ranked loopless paths. Another known algorithm is by Eppstein 1998, which allows loops and has better performance. However, one drawback of Eppstein's algorithm is that it requires the graph to be fully defined and available in memory. The recent extension of the Eppstein's algorithm called the K^* algorithm (Aljazzar and Leue 2011) overcomes this problem by supporting on-the-fly construction of the graph and thus allows use of heuristic search, making it a very strong candidate match for planning problems.

In this paper, we introduce and compare four planning algorithms for top-k planning: iterative replanning using LAMA or other existing high-performance planners, branch and bound, a planning algorithm based on the Eppstein's k shortest paths algorithm, and an algorithm based on the K^* algorithm. We call the top-k planner based on Eppstein's algorithm, *TK*, and the planner based on the K^* algorithm, *TK**. Note, that our algorithms are based on known existing methods, but are employed to address the top-k planning problem. Our experiments show that planning time required for top-k planning is comparable to cost-optimal planning that finds a single cost-optimal plan. We also find that *TK** outperforms all other approaches by a large margin.

Problem Formulation

Top-k planning problem is defined as $R' = (F, A, I, \mathcal{G}, k)$, where F is a finite set of fluent symbols, A is a set of actions with non-negative costs, I is a clause over F defining the initial state, $\mathcal{G}$ is a clause over F defining the goal state, and k is the number of plans to find. The set of plans $\pi = \{\alpha_1, ..., \alpha_k\}$ is the solution to the top-k planning problem R' if an only if each plan $\alpha_i \in \pi$ is a solution to the cost-optimal planning problem $(F, A, I, \mathcal{G})$ and there does not exists a plan α' for $(F, A, I, \mathcal{G})$, $\alpha' \notin \pi$ such that $cost(\alpha') < cost(\alpha_i)$ for all $\alpha_i \in \pi$. It follows that at least one optimal plan is in the set of plans π if $k > 0$.

Note, while we indicated that the goal state, $\mathcal{G}$, is in a form of a final-state goal in the definition of R', we consider temporally extended goals as well. Temporally extended goals such as sequence of observations from a system description either totally ordered or partially ordered can be compiled away to final-state goal following a compilation technique discussed in several papers (e.g., (Sohrabi, Baier, and McIlraith 2010; Haslum and Grastien 2011)); the temporally extended goals can be compiled away by an action which enforces the temporal sequence of the goal.

Top-k Planning via Iterative Replanning

The first of the four approaches we describe builds upon existing Planning Domain Definition Language (PDDL) (McDermott 1998) planners, extending the applicability of those planners to top-k planning problems. We have introduced this approach in prior work as a simple practical solution for top-k problems (Sohrabi, Udrea, and Riabov 2013). In our experiments we used LAMA (Richter and Westphal 2010), as one of the fastest planners available, but the approach does not depend on the choice of the planner. LAMA can be used to find cost-optimal plans by modeling costs as domain variables, and the last returned plan is optimal if LAMA is given sufficient time to complete the search and exit (this was confirmed in our experiments).

The main idea is to use a PDDL planner iteratively, slightly modifying the problem each time, until top-k plans are found. To solve the top-k planning problem $R' = (F, A, I, \mathcal{G}, k)$, we solve a sequence of cost-optimal planning problems $R_i = (F_i, A_i, I_i, \mathcal{G}_i)$, starting with finding the optimal plan of length n for $R_1 = (F, A, I, \mathcal{G})$. Then, given a cost-optimal plan p for R_i, and assuming m problems were created at a previous iteration, we create a new set of problems $\{R_j | j = m+1, ..., m+n)\}$ by modifying R_i by adding, for each action that occurs in p, a new precondition that prevents that action from appearing at the same position in the new plan. The best solution to the new problems R_j will be used to find the next-best plan p'. The generated problems are modified again to generate new problems and find the next best plan, until k such plans are found. Note that this algorithm does not find plans that have top-k plans as their prefixes (i.e., plans that reach the goal more than once). Overall, solving the top-k planning problem requires at most $O(N^k)$ replanning iterations, where N is the length of the longest plan among the top-k plans.

The actions are modified by introducing new predicates *(at-pos ?i)* and *(next ?i ?j)* to keep track of the position of each action in the plan. For example, the initial state will include predicates *(at-pos p1) (next p1 p2), (next p2 p3)*, etc., and for each action the precondition will include *(at-pos ?i) (next ?i ?j)*, while the effect of the same action will include *(at-pos ?j) (not (at-pos ?i))*. This modification does not change the set of valid plans. However, it allows disabling application of an action by adding a negated precondition: for example, adding *(not (at-pos p3))* to an action will prevent that action from appearing at the third position in any valid plan.

The outline of this algorithm is presented below.

```
0. Find plan α for the original problem R.
1. Set π = {α}.
2. Add each action a of α and its position i
    S = {(a,i)} to future exploration list L.
3. For each S in L
4.    For each (a,i) ∈ S
5.         Add negated predicate associated
           with a,i to action a.
6.    Generate a plan α' for the new problem
      where all actions in S are disallowed.
7.    For each action a at position i in α'
8.         Add the set S ∪ {(a,i)} to L'.
9. Add one of the plans α' with minimum cost to π.
10. Replace L with L'.
11. If |π| < k and L ≠ ∅ go to step 2.
12. Return π as the solution to the problem.
```

Top-k Planning via Branch and Bound

The second approach we propose for top-k planning problem is based on branch and bound. Unlike iterative replanning, it does not require solving many similar planning problems. Branch and bound is a general framework used for finding optimal solutions in a variety of settings, and it can be modified for solving top-k problems. The additional advantage of the general framework is the flexibility it allows in defining optimization objectives and constraints.

Branch and bound begins by selecting a variable for branching (e.g., the variable can represent the action applied at the first step of the plan), creates a search node for every feasible value of the variable, and computes lower and upper bounds on the possible solutions. For example, the lower bound can be the cost of the selected action and the upper bound can be infinity. Assuming that a minimization problem is being solved, search nodes can be pruned if their lower bound value exceeds already known upper bound on the solution, or the value of the current best solution. In the next iteration, which can be done separately for each search node, the next variable is selected, and the next set of search nodes is created following the same procedure.

This standard framework can be modified to find top-k solutions instead of a single optimal solution, by pruning the search tree based on the k-th best solution found instead of using the value of the best solution. When search terminates, the remaining k best solutions will be the top k solutions. In most scenarios, however, this modification increases the search time, especially for large values of k, because the search tree cannot be efficiently pruned until k complete solutions are found.

In our implementation we apply forward branching on operators of the planning problem after grounding. Grounding, implemented during preprocessing, assigns fixed values to variables of actions and thus creates multiple operators from a single action. During this preprocessing, an index of operators is also created, based on the matching between preconditions and effects. To control the potential exponential explosion of the number operators, we implement grounding based on forward reachability within a relaxed planning problem without deletes. We note that despite the reduction in the number operators thanks to reachability analysis, a significant fraction of operators created during grounding may still never be explored during search, since action costs are not accounted for in this process. The fourth approach we describe further in this paper will improve on this by implementing dynamic grounding.

The final algorithm can be summarized as follows.

```
0. Read planning problem R' = (F, A, I, G, k);
   Set U = {}, π = {}; Set UB = ∞;
   Insert a partial plan α = {I} into U.
1. Apply forward grounding to A
   to create operator set O.
2. If U is empty, return the set of plans π.
3. Remove a partial plan α from U.
4. If for last state s of α, s ∈ G Then
5.   Set π = π ∪ {s}.
6.   If |π| > k Then
7.      Set π = π \ arg max{cost(α') | α' ∈ π}.
8.      If |π| = k Then
9.         Set UB = max{cost(α') | α' ∈ π}.
10. Find operators {o} ⊂ O applicable in s;
    Set U = U ∪ {α' = (α, o) | cost(α') < UB}.
11. Repeat from step 2.
```

Note the algorithm assumes that all actions have nonnegative costs, and therefore costs of partial plans can be used as lower bounds for complete plans in Step 10. In addition, a variety of heuristics can be used to order the set of partial plans U in Step 3, and the choice of the heuristic will affect performance. In our implementation we sort partial plans by distance to goal computed based on the relaxed formulation used during grounding. Finally, while we have not done this ourselves, branch and bound and heuristic search share multiple features, and modifications for finding top-k plans can similarly be made to heuristic search algorithms.

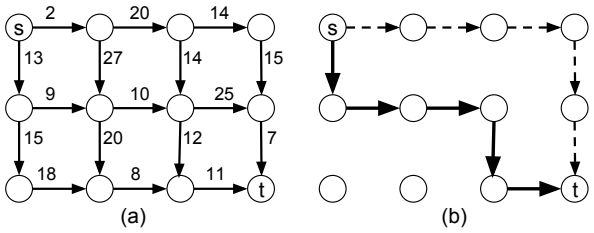


Figure 1: (a) shows the nodes and edges of a graph with source node s and terminal node t with edge lengths specified on the edges; (b) shows the shortest path in bold arrows and the second shortest path in dashed arrows.

Top-k Planning via K Shortest Paths

The cost-optimal planning problem can be modeled as the problem of finding the shortest path in state space from initial state to the goal. In this section we build on this idea by applying Eppstein's k shortest paths algorithm (Eppstein 1998) in state space to solve the top-k planning problem. The resulting algorithm is very efficient, but requires the complete graph of states and actions to be available in memory. Constructing this graph is expensive in large problems, and this shortcoming will be addressed using an improved variant of the algorithm in the approach described in the next section. In this section we first introduce notation for the k shortest paths problem, and then describe the planning algorithm based on Eppstein's k shortest paths.

Background: K Shortest Paths Problem

K shortest paths problem is an extension of the shortest path problem where in addition of finding one shortest path, we need to find a set of paths that represent the k shortest paths (Hoffman and Pavley 1959). Following Eppstein 1998, k shortest path problem is defined as 4-tuple $R = (G, s, t, k)$, where $G = (V, E)$ is a graph with a finite set of n nodes (or vertices) V and a finite set of m edges E, s is the source node, t is the destination node, and k is the number of shortest paths to find. Each edge $e \in E$ has a *length* (or *weight* or *cost*), which we denote by $l(e)$. The length of a path p, $l(p)$, is consequently defined by the sum of its edge lengths. The *distance* $d(u, v)$ for any pair of nodes u and $v \in V$ is the length of the shortest path between the two nodes. Hence, $d(s, t)$ is the length of the shortest path for the problem R. Figure 1 shows an example from (Eppstein 1998) to illustrate the terminology. The distance $d(s, t) = 55$, is the length of the shortest path shown in bold; the length of the second shortest path is 58.

The set of paths $P = \{p_1, p_2, ..., p_k\}$ is the solution to the k shortest paths problem R if and only if it is a set of shortest paths from node s to node t. That is each $p_i \in P$, $1 \leq i \leq k$, is a path in graph G and there does not exists a path p' in graph G, $p' \notin P$ such that $l(p') < l(p_i)$ for all $p_i \in P$. That is, there is no path, except amongst the k

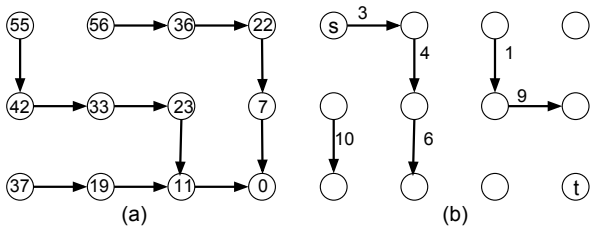


Figure 2: (a) shows the shortest path tree T and distance to destination t; (b) shows the side edges with their associated detour cost.

shortest paths, with better length than any of the paths in the set P. It follows that at least one shortest path with length $d(s,t)$ is in the set P if $k > 0$.

Background: Eppstein's Algorithm (EA)

Given a k shortest paths problem $R = (G, s, t, k)$, the EA algorithm first computes a single-destination shortest path tree with t as the destination (or the reversed single-source shortest path tree) by applying Dijkstra's algorithm on G. The edges in the resulting *shortest path* tree, T are called the *tree* edges while all the missing edges (i.e., the edges in $G - T$) are called the *sidetrack* edges. Each edge in G is assigned a number that measure the detour cost of taking that edge. Consequently, the detour cost of the tree edges is 0, while the detour cost of the sidetrack edges is greater than 0. Figure 2 shows the shortest path tree T and the side edges along with their detour cost of our earlier example.

The EA algorithm then constructs a complex data structure called *path graph* $P(G)$ that stores the all paths in G, where each node in represents a sidetrack edge. This is followed by the use of Dijkstra search to $P(G)$ to extract the k shortest paths. An important property is that given a sequence of sidetrack edges representing a path in $P(G)$ and the shortest path tree T, it is possible to uniquely construct a s-t path in G. This can be done by using sub-paths from T to connect the endpoints of sidetrack edges. Given this property and the special structure of $P(G)$, it is ensured that the i-th shortest path in $P(G)$ results in a sidetrack sequence which can be mapped to the i-th shortest path in G. By construction, $P(G)$ provides a heap-ordered enumeration of all paths in G, and since every node of $P(G)$ has limited outdegree (at most 4), the complexity of enumerating paths in increasing cost order is bounded. The worst-case runtime complexity of the EA algorithm is $O(m + n \log n + kn)$. This complexity bound depends on a compact representation of the resulting k paths, and can be exceeded if the paths are written explicitly, by enumerating all nodes and links, as we have done in our planner implementation. For more details see (Eppstein 1998).

Top-k Planning Algorithm Based on EA

Our planning algorithm can be summarized as follows. We call the top-k planner based on this algorithm, *TK*.

```
0. Read planning problem R' = (F, A, I, G, k).
1. Apply forward grounding to A
   to create operator set O.
2. Initialize G=(V,E): let V={I}, E=∅.
3. Let U={I}.
4. For each state s ∈ U
5.   U = U − {s}
6.   For each operator o ∈ O
     such that s satisfies precondition of o
7.       Let s' = o(s).
8.       If edge o(s,s') ∉ E Then
9.           If s' ∉ V Then
10.              Let V = V ∪ {s'}, U = U ∪ {s'}.
11.          Add o(s,s') to E.
12.          Let cost(o(s,s')) = cost(o).
13. If U ≠ ∅ goto step 4.
14. Apply EA to G to find k shortest paths.
```

This algorithm consists of three main stages. Step 1 implements action grounding. Steps 2-12 implement forward search to construct the complete state transition graph G. Finally, step 13 applies Eppstein's algorithm to the resulting graph. Since nodes in G represent states and edges in G correspond to operators, all paths in G correspond to plans in R', and paths have the same cost as corresponding plans. Therefore, the solution produced by Eppstein's algorithm can be directly used as a solution to the top-k planning problem. We note that in our experiments the first two stages, grounding and creating the state graph, taken together, took approximately the same amount of time as the last stage.

Top-k Planning via K^* Search

The major bottleneck of the previous approach is the construction of the complete state transition graph, which may include a huge number of states that are very far away from the goal, and would not appear in top-k plans. Planners commonly deal with this challenge by relying on heuristic search algorithms like A* to dynamically expand only the necessary portion of the state graph during search, while being guided by a heuristic toward the goal (e.g., FF (Hoffmann and Nebel 2001) and Fast Downward (Helmert 2006)), and the effectiveness of this approach has been proven (Bonet and Geffner 2001). The $K*$ algorithm proposed by Aljazzar and Leue combines the best of both worlds: it allows constructing the graph G dynamically using heuristic-guided A* search, while updating its equivalent of $P(G)$ to find k shortest paths. In addition to eliminating the complete state graph construction, with K^* we can ground actions dynamically, eliminating the expensive grounding stage.

Background: K^* Algorithm

The K^* algorithm (Aljazzar and Leue 2011) uses many of the same concepts as in the EA algorithm including sidetrack edges, detour costs, and the path graph $P(G)$ (although with a few differences in its construction) and has the same worst-case complexity as the EA algorithm. However the K^* algorithm has better performance in practice because unlike the EA algorithm it does not require the graph G to be completely defined or available when the search starts. It also does not perform the all-nodes shortest path computation on G to compute the shortest path tree T. In short, the K^* algorithm works as follows. The first step is to apply a forward A* search to construct a portion of graph G. The second step is suspending A* search, updating $P(G)$ to include nodes and sidetracks discovered by A*, and applying Dijkstra to $P(G)$ to extract solution paths and resuming the A* search. The use of A* search to dynamically expand G enables the

use of heuristic search and also allows extraction of the solution paths before G is fully explored.

Top-k Planning Algorithm Based on K^*

In the implementation of the planning algorithm we follow the algorithm structure imposed by K^*, as follows. Note that we call our top-k planner that is based on K^*, ***TK******.

```
0. Read planning problem R' = (F, A, I, G, k).
1. Expand the state graph G by using A*
   and applying actions to compatible states
   starting from I, and until G is reached.
2. Continue applying A* to expand G
   until 20% increase in links or nodes.
3. Update P(G) based on new links in G.
4. Apply Dijkstra step
   to extract the next path from P(G).
5. If k paths are found
6.      Exit.
7. If K* scheduling condition is reached
8.      Goto step 2.
9. Goto step 4.
```

The K^* scheduling condition is evaluated by comparing the state of A* and Dijkstra searches, as defined in K^* algorithm. It determines whether new links must be added to G before resuming Dijkstra search on updated $P(G)$. There is no separate grounding stage, since actions are ground at the same time when they are applied during A* search. The amount of A* expansion required before resuming Dijkstra (in our implementation, 20%), is an efficiency tradeoff, and 20% is the same value that was used in experiments in the original K^* paper (Aljazzar and Leue 2011). Of course, step 2 may also be completed if no new links can be added.

Overall, due to multiple improvements in efficiency made possible by this algorithm, ***TK****** was the best performing in our experiments. We also expect that with some work this approach can be integrated into planners that use A* search, enabling those planners to solve top-k problems.

In our experiments, ***TK****** with constant 0 heuristic performs very well, and we have not experimented with other, potentially better performing heuristics. This is an interesting direction for improvement that could be explored in future work. Even though this is not a requirement for K^* in general, our implementation requires a consistent heuristic, which did not allow us to experiment with, for example, lookahead heuristics. Further, the dynamic grounding prevented the use of heuristics used in the Branch and Bound approach, since those heuristics require static grounding.

Experimental Evaluation

In this paper we argue that practical solutions can be developed for the top-k planning problem. To that end, we have 3 main objectives in our experiments. First, we measure the change in performance that results from the requirement to find top-k plans instead of a single cost-optimal plan. Second, we compare the performance of four different approaches we propose. Third, we measure the effect that increasing the value of k will have on planning time.

Generated Random Problem Instances

The approach we introduced in this paper is general and can be applied in a variety of applications that require cost-optimal planning. As a benchmark for performance evaluation, we have generated random instances of varying size based on the hypothesis exploration problem with unreliable observations (Sohrabi, Udrea, and Riabov 2013). This application provides a good example of a challenging top-k planning problem, and generated problems typically have a very large number of possible plans with different costs. The domain and the generated problems were represented in a STRIPS-like planning language recognized by our planner, as well as in PDDL for LAMA.

All generated problems share a planning domain description containing 6 actions and 8 predicates. In this domain, low costs were assigned to actions used in perfect explanations of observations, and high costs to actions representing exceptions, such as unexplained observations or state transitions without observations. To generate a random problem instance, we generated a random state transition system with a given number of states.

Malware Detection Instances

In addition to randomly generated state transition systems, we used the malware detection problem (18 states), as described in (Sohrabi, Udrea, and Riabov 2013). In short, the malware detection problem involves generating hypotheses about the network hosts by analyzing the network traffic data. To make this possible, the domain description includes the states of the host (e.g., infected with malware due to downloading an executable file or the *Command & Control Rendezvous* state via Internet Relay Chat (IRC)) and transitions between these states and many-to-many correspondence between states and observations. The results for this domain is shown under the "Malware Domain" rows.

Planning Time for The Top-k Problem

We have varied the size of the problem by changing the number of the states of the system being modeled (not to be confused with planning states) and the number of observations received from the system. For all time measurements in this paper we used the same Quad-core 2.93 GHz Intel Xeon X5570 processor with 32 GB RAM and 64-bit Red-Hat Linux OS.

Table 1 presents the results of comparison between approaches described in this paper. For all algorithms except iterative replanning, we measured time it took to find top $k = 50$ plans. For iterative replanning ("LAMA top-1" column in the table), we measured the duration of a single iteration, i.e., solving one cost-optimal planning problem using LAMA, while at least $k = 50$ iterations will be required for the top-k problem (and in the worst case, exponentially more). Conveniently, this also helps compare planning time of top-k and regular cost-optimal planning. During measurement we enforced a limit on planning time of 300 seconds, and the instances where this limit was exceeded are indicated by "-" in the table. For LAMA the time we report is the time that it took for LAMA to terminate (i.e., exhaust the search space) and hence its last returned plan is cost-optimal or top-1. The "-" entries indicate that LAMA was not able to find the cost-optimal plan or terminate its search before the time limit is reached.

	LAMA, top-1			Branch & Bound, top-50			*TK*, top-50			*TK* *, top-50		
Problem size	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
Malware Domain, 5 obs.	0.52	0.64	1.16	0.11	0.22	0.39	0.06	0.07	0.09	0.04	0.06	0.08
10 states, 5 obs.	0.35	0.52	1.68	0.08	0.13	0.22	0.05	0.05	0.06	0.03	0.04	0.06
50 states, 5 obs.	0.94	1.08	1.17	0.36	0.56	0.97	0.18	0.21	0.24	0.06	0.08	0.10
100 states, 5 obs.	2.64	2.95	3.20	1.45	2.49	4.19	1.03	1.23	1.47	0.18	0.22	0.25
Malware Domain, 10 obs.	0.69	0.75	0.86	0.28	3.67	10.33	0.09	0.12	0.15	0.06	0.07	0.11
10 states, 10 obs.	0.40	0.47	0.54	0.64	0.97	1.59	0.06	0.07	0.08	0.04	0.06	0.07
50 states, 10 obs.	1.50	1.89	2.37	1.46	7.86	37.04	0.42	0.51	0.60	0.10	0.12	0.13
100 states, 10 obs.	5.20	6.27	9.14	5.88	21.08	51.93	2.62	3.43	3.97	0.21	0.35	0.55
Malware Domain, 20 obs.	1.06	1.37	2.10	1.49	-	-	0.19	0.27	0.33	0.08	0.12	0.38
10 states, 20 obs.	0.50	0.71	0.94	7.24	31.13	132.34	0.10	0.13	0.15	0.06	0.09	0.37
50 states, 20 obs.	3.50	4.48	6.65	11.20	77.11	300.08	1.36	1.72	2.02	0.17	0.21	0.30
100 states, 20 obs.	12.08	20.11	28.01	51.14	-	-	8.52	10.55	12.10	0.46	0.66	0.82
Malware Domain, 60 obs.	2.93	4.30	7.04	25.05	-	-	1.06	1.45	2.15	0.08	0.15	0.23
10 states, 60 obs.	1.98	2.65	3.22	-	-	-	0.44	0.54	0.70	0.14	0.17	0.20
50 states, 60 obs.	18.93	61.96	134.84	-	-	-	9.49	12.52	15.48	0.35	0.60	0.80
100 states, 60 obs.	107.39	-	-	-	-	-	56.52	75.17	102.63	1.12	2.07	2.81
Malware Domain, 120 obs.	6.63	10.23	16.93	-	-	-	4.61	5.77	8.94	0.15	0.26	0.47
10 states, 120 obs.	5.83	9.28	22.23	-	-	-	1.81	2.40	3.27	0.27	0.33	0.41
50 states, 120 obs.	69.98	-	-	-	-	-	40.42	51.91	70.16	0.90	1.51	2.23
100 states, 120 obs.	-	-	-	-	-	-	229.22	294.80	-	2.81	5.35	7.68

Table 1: Relative performance: Minimum, maximum and average planning time, in seconds, for 15 instances of each size.

The results were obtained on the same problem instances, and help illustrate the advantages and disadvantages of the algorithms we evaluated. While iterative replanning is the easiest to implement and may perform well in practice on small instances, it is by far the worst performing, as expected. Iterative replanning results were omitted from Table 1 to save space, but they can be easily estimated based on "LAMA top-1", by multiplying the time of one iteration by a very optimistic estimate of the minimum number of iterations (in this case, 50). Branch and bound generalizes for a variety constraints and objective functions, but as expected, it is not as fast as specialized shortest paths algorithms.

The unexpected result is how well *TK* and *TK** perform, in comparison with the time it takes LAMA to find a single plan. While our implementation of the Eppstein's algorithm, very fast on small problems, is limited on large problems by the requirement to create the complete planning state graph, *TK** does not have that limitation, and performs much better. For example, for the largest problem size of 100 system states and 120 observations, *TK** is on average 55 times faster by the next fastest approach, *TK*.

We note that *TK** is the only approach that implements dynamic grounding. In our experiments action grounding is responsible for roughly half of the planning time of k shortest paths algorithm, and since the same grounding implementation is used for branch and bound, it suffers the same performance penalty.

The Impact of the Value of K

Above, we showed that *TK* and *TK** scale well with increasing problem size. Notably, these planners also scale well with increasing k. To measure this, we rerun the same experiments with k=1000, and these results, along with the results for k=50 from Table 1, for the two best planning approaches, are presented in Table 2.

As previously shown for *TK* and *TK**, while *TK** is faster overall, it is more sensitive to the value of k, and larger values lead to somewhat longer planning times. For example, for the largest problem of 100 states and 120 observations, for k=1000 the average planning time increases by 70% compared to k=50. *TK* spends significant time upfront computing a shortest path tree covering the entire state graph, but finding individual plans after that is very fast, and difference in planning time between k=50 and k=1000 for *TK* is negligible. For the same problem size, average planning time increases only by approximately 1%.

Conclusions

Our work on top-k planning is motivated by a specific application where finding multiple high-quality plans is required, namely hypothesis exploration for malware detection (Sohrabi, Udrea, and Riabov 2013). We proposed a new top-k plans formulation for cost-optimal planning, which can be used in this and other applications.

Generating diverse plans is a notable related work (e.g., (Myers and Lee 1999; Srivastava et al. 2007; Nguyen et al. 2012)). However, rather finding a representative set of plans, our approach focuses on computing top-k plans. Also, generating Pareto frontiers or a Pareto set (e.g., (Sroka and Long 2012; Khouadjia et al. 2013)) is related. Like diverse plans, this work does not focus on finding all top-k plans in any of the dimensions of the objective function, instead multiple diverse plans from a Pareto optimal curve are found. Furthermore, we do not rely on additional objectives to find near-optimal plans with a single objective.

We have implemented and evaluated four top-k planning algorithms, two based on the k shortest paths algorithm by Eppstein and its successor K^*. We also compared the result of these algorithms with the result of computing top-k plans from our iterative replanning and branch and bound algorithms. The results show that computation of top-k plans is comparable to the computation of a single cost-optimal plan. Additionally, we found that our top-k planning system based on the K^* algorithm, *TK**, is as expected, the most promis-

	TK, top-50			*TK*, top-1000			*TK* *, top-50			*TK* *, top-1000		
Problem size	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
10 states, 5 obs.	0.05	0.05	0.06	0.27	0.29	0.35	0.03	0.04	0.06	0.21	0.24	0.28
50 states, 5 obs.	0.18	0.21	0.24	0.42	0.45	0.48	0.06	0.08	0.10	0.24	0.28	0.41
100 states, 5 obs.	1.03	1.23	1.47	1.30	1.47	1.74	0.18	0.22	0.25	0.41	0.44	0.47
10 states, 10 obs.	0.06	0.07	0.08	0.42	0.49	0.53	0.04	0.06	0.07	0.32	0.38	0.42
50 states, 10 obs.	0.42	0.51	0.60	0.83	0.94	1.10	0.10	0.12	0.13	0.43	0.48	0.53
100 states, 10 obs.	2.62	3.43	3.97	3.09	3.85	4.44	0.21	0.35	0.55	0.67	0.75	0.81
10 states, 20 obs.	0.10	0.13	0.15	0.90	0.96	1.03	0.06	0.09	0.37	0.72	0.74	0.78
50 states, 20 obs.	1.36	1.72	2.02	2.23	2.56	2.85	0.17	0.21	0.30	0.81	0.89	0.93
100 states, 20 obs.	8.52	10.55	12.10	9.33	11.44	12.96	0.46	0.66	0.82	1.10	1.36	1.49
10 states, 60 obs.	0.44	0.54	0.70	2.69	2.95	3.34	0.14	0.17	0.20	1.98	2.10	2.24
50 states, 60 obs.	9.49	12.52	15.48	11.82	14.95	18.33	0.35	0.60	0.80	2.24	2.52	2.75
100 states, 60 obs.	56.52	75.17	102.63	57.93	77.72	106.43	1.12	2.07	2.81	2.96	4.07	4.73
10 states, 120 obs.	1.81	2.40	3.27	6.58	7.18	7.84	0.27	0.33	0.41	4.07	4.24	4.44
50 states, 120 obs.	40.42	51.91	70.16	45.92	57.22	75.92	0.90	1.51	2.23	4.67	5.40	6.01
100 states, 120 obs.	229.22	294.80	412.89	234.58	300.41	419.97	2.81	5.35	7.68	6.55	9.13	11.37

Table 2: The impact of the value of k: Minimum, maximum and average planning time, in seconds, for 15 instances of each size.

ing direction for top-k planners, and in our implementation it performed more than 100 times faster than all other algorithms (in part due to faster grounding). To conclude, the contribution of this paper is: 1) the formulation of the top-k planning problem and its reduction to a k shortest paths computation in a graph, 2) comparison of four implementations of the top-k planner, 3) experimental evaluation of performance of our implementations on synthetic benchmarks derived from a real-world scenario.

References

Aljazzar, H., and Leue, S. 2011. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence* 175(18):2129–2154.

Baier, J., and McIlraith, S. 2008. Planning with preferences. *AI Magazine* 29(4):25–36.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Botea, A., and Harabor, D. 2013. Path planning with compressed all-pairs shortest paths data. In *Proc. of the 23rd Int. Conference on Automated Planning and Scheduling (ICAPS)*, 293–297.

Botea, A. 2011. Ultra-fast optimal pathfinding without runtime search. In *Proc. of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 112–127.

Edelkamp, S., and Kissmann, P. 2009. Optimal symbolic planning with action costs and preferences. In *Proc. of the 21st Int. Joint Conference on Artificial Intelligence (IJCAI)*, 1690–1695.

Eppstein, D. 1998. Finding the k shortest paths. *SIAM Journal on Computing* 28(2):652–673.

Haslum, P., and Grastien, A. 2011. Diagnosis as planning: Two case studies. In *Int. Scheduling and Planning Applications woRKshop (SPARK)*, 27–44.

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

Hoffman, W., and Pavley, R. 1959. A method for the solution of the *n*th best path problem. *Journal of the ACM* 6(4):506–514.

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

Kambhampati, S. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proc. of the 22nd National Conference on Artificial Intelligence (AAAI)*, 1601–1604.

Keyder, E., and Geffner, H. 2009. Soft Goals Can Be Compiled Away. *Journal of Artificial Intelligence Research* 36:547–556.

Khouadjia, M. R.; Schoenauer, M.; Vidal, V.; Dréo, J.; and Savéant, P. 2013. Pareto-based multiobjective AI planning. In *Proc. of the 23rd Int. Joint Conference on Artificial Intelligence (IJCAI)*, 2321–2327.

McDermott, D. V. 1998. PDDL — The Planning Domain Definition Language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Myers, K. L., and Lee, T. J. 1999. Generating qualitatively different plans through metatheoretic biases biases. In *Proc. of the 16th National Conference on Artificial Intelligence (AAAI)*, 570–576.

Nguyen, T. A.; Do, M. B.; Gerevini, A.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence* 190:1–31.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proc. of the 21st Int. Joint Conference on Artificial Intelligence (IJCAI)*, 1778–1783.

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

Sohrabi, S.; Baier, J.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *Proc. of the 12th Int. Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 26–36.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2011. Preferred explanations: Theory and generation via planning. In *Proc. of the 25th National Conference on Artificial Intelligence (AAAI)*, 261–267. Accepted as both oral and poster presentation.

Sohrabi, S.; Udrea, O.; and Riabov, A. 2013. Hypothesis exploration for malware detection using planning. In *Proc. of the 27th National Conference on Artificial Intelligence (AAAI)*, 883–889.

Srivastava, B.; Nguyen, T. A.; Gerevini, A.; Kambhampati, S.; Do, M. B.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *Proc. of the 20th Int. Joint Conference on Artificial Intelligence (IJCAI)*, 2016–2022.

Sroka, M., and Long, D. 2012. Exploring metric sensitivity of planners for generation of pareto frontiers. In *Proc. of the 6th Starting AI Researchers' Symposium (STAIRS)*, 306–317.

Yen, J. 1971. Finding the k shortest loopless paths in a network. *Management Science* 17:712–716.

Zhu, A. D.; Ma, H.; Xiao, X.; Luo, S.; Tang, Y.; and Zhou, S. 2013. Shortest path and distance queries on road networks: towards bridging theory and practice. In *ACM SIGMOD Int. Conference on Management of Data (SIGMOD)*, 857–868.

The Application of Planning to Urban Traffic Control

Falilat Jimoh and **Lukáš Chrpa** and **Thomas Leo McCluskey**
Department of Informatics
University of Huddersfield
United Kingdom

Abstract

In this position paper we discuss our current work that aims to develop a technology based on Automated Planning applied to handling unforeseen situations in urban traffic control. To achieve this, we advocate the need for planners which can reason with mixed discrete/continuous variables in dynamic environments.

Introduction

Control systems which support urban traffic control (UTC), such as those controlling networks of traffic lights, have utilised AI techniques since the 1970's. These systems are embedded in a real time control environment, and are often based on algorithms that rely on feedback and adaptation. They use road traffic data which may be current (gathered every few seconds) or historic (gathered‘ over several years). For instance, current traffic control systems often operate on the basis of adaptive green phases and flexible co-ordination in road (sub) networks based on measured traffic conditions.

These approaches, however, are still not very efficient during unforeseen situations such as road incidents when changes in traffic are requested in a short time interval (Roozemond 2001; De Oliveira and Bazzan 2009). In such circumstances, traffic control systems usually use fixed traffic signal timing or apply some hard-coded approach in order to revert back into a recognized state.

Therefore, we need systems that can plan and act effectively in order to restore an unexpected road traffic situation into a normal order. One promising direction is in creating a generic architecture that enables control systems to automatically reason with knowledge of their environment and their controls, in order to generate plans and schedules from first principles to manage themselves in unforeseen situations. This would be a significant step forward in the urban traffic control. A step towards this would be to exploit Automated Planning techniques which can reason about unforeseen situations in the road network and generate plans (sequences of actions) achieving a desired traffic situation. In fact we see AI planning as having a vital role to play in achieving such kind of robust control system, and this paper proposes such potential in the domain of urban traffic control.

To improve reasoning with continuously changing variables with respect to time in the urban traffic environment, it is necessary to incorporate knowledge of control processes into planning engines. We aim to develop a planner which supports creating and analysing domain descriptions and plans containing continuous processes, events and actions.

Problem Definition and Related Work

We are interested in such domains that are modelled by using variables which are changing continuously, as parts of processes and events that are both internal and exogenous. Most existing planners can only reason with classical domain models, i.e., deterministic, fully observable and static environments, though planning engines which can reason with mixed discrete/continuous domains have been recently developed (Coles et al. 2008; Ono, Williams, and Blackmore 2013; Coles and Coles. 2013). Shin and Davis (2005) use a compilation of SAT and linear programming techniques to solve planning problems with numeric variables allowing linear continuous change. UPMurphi (Penna et al. 2009), on the other hand, handles with both linear and non-linear continuous changes, but it has the limitation depending on a hand-crafted discretisation of time to enable reasoning with continuous change.

An interesting approach to fuse plan generation and control in a continuous system is to use an external module connected to a planner (Lhr et al. 2012; Piacentini et al. 2013). This helps to control the overall dynamics of a continuous system by minimising the deviation of numeric state variables continuously changing over the time from their desirable values. This approach is used to overcome weaknesses in existing planning engines in order to cope with continuously changing state variables. However, there is a need to design and develop planning engines that can reason with continuous processes and generate optimal or nearly optimal plans in reasonable time.This also calls for the development of new heuristics for such problems considering the complexity of the numeric variables involved, for instance, in urban traffic processes.

In the applications we consider (e.g. Urban Traffic Control), we need a planning technology that can:

- reason with the system model as well as the disturbances in the system (disturbances in a system are unforeseen changes to the state variables)
- generate a plan irrespective of whether the goal is fully or partially satisfied
- exhibit the "anytime" planning property
- deal with multi-variable constraints in the presence of a fixed or continuous changing time-stamp.

Modelling this problem requires a language that can represent mixed discrete / continuously changing variables, such as $PDDL^+$(Fox and Long 2006). Although $PDDL^+$ provides increased expressiveness, with the representation of continuous processes and events in domain models, we need more expressiveness for the traffic domain beyond existing description languages. Thus we will modify existing tools by using traffic representation and reasoning schema in our domain and problem description such as multi-variable constraints.

Our Research Program

In our previous work, we have introduced the problem of self-management of a road traffic network as a temporal planning problem in order to effectively navigate cars throughout a road network in urban areas (Jimoh et al. 2013). So, we introduced Automated Planning into UTC which, moreover, can re-route traffic flow when a road becomes unavailable due to unexpected circumstances. As part of this effort, we embedded the knowledge of UTC structure into a planning domain model and evaluated the possibility of reasoning with this knowledge and optimising traffic flow in situations where a given road within a network of roads becomes unavailable due to unexpected situations such as road accidents. This allows us to control traffic more efficiently in the road network. Our preliminary experimental evaluation showed that our approach is able to provide plans in a reasonable time.

Embedding a planning component consisting of state-of-the-art domain-independent planning engines into our urban traffic environment, does not provide optimal solutions and, moreover, solutions are often very sub-optimal even in quite simple cases. Also, it is questionable whether planners' performance would not significantly decrease on larger road networks. On the other hand, developing a domain-dependent planner specifically tailored to urban traffic control might overcome (some of) these issues. We need to be able to reason with discrete (logical) and continuous (numeric) state variables as well as to reason with discrete and continuous time change in the UTC domain. Existence of an efficient domain-dependent continuous planner which can also handle continuous processes and events will be sufficient for the UTC domain to demonstrate usefulness of AI planning there. This type of problem in flow processing is presently optimised with Model Predictive Control (MPC) strategy. Thus, exploiting such a control approach and embedding it into existing state-of-the-art planners would lead to an improvement in planning with continuous variables.

Our present efforts are focused to using a MPC architecture(Al-Gherwi, Budman, and Elkamel 2011; Veselý, Rosinov, and Foltin 2010; Camacho and Bordons 1999) to implement a continuous planner which can generate plans that will get and keep the urban traffic controller in a desirable state in order to optimize traffic flows in urban areas.

References

Al-Gherwi, W.; Budman, H.; and Elkamel, A. 2011. A robust distributed model predictive control algorithm. *Journal of Process Control* 21(8):1127–1137.

Camacho, E., and Bordons, C. 1999. *Model predictive control*. London: Springer.

Coles, A. J., and Coles., A. I. 2013. Pddl+ planning with events and linear processes. In *Proceedings of the 1st Workshop on Planning in Continuous Domains at the Twenty Third International Conference on Automated Planning and Scheduling (ICAPS-13)*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2008. Planning with linear continuous numeric change. In *Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2008)*.

De Oliveira, D., and Bazzan, A. L. C. 2009. *Multiagent Learning on Traffic Lights Control: Effects of Using Shared Information*. 307–322.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *J. Art. Int. Res. (JAIR)* 27:235–297.

Jimoh, F.; Chrpa, L.; McCluskey, T.; and Shah, M. M. S. 2013. Towards application of automated planning in urban traffic control. In *16th International IEEE Conference on Intelligent Transportation Systems*. automated planning, urban transport control, autonomic systems.

Lhr, J.; Eyerich, P.; Keller, T.; and Nebel, B. 2012. A planning based framework for controlling hybrid systems.

Ono, M.; Williams, B. C.; and Blackmore, L. 2013. Probabilistic planning for continuous dynamic systems under bounded risk. *J. Artif. Int. Res.* 46(1):511–577.

Penna, G. D.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. Upmurphi: A tool for universal planning on pddl+ problems. In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *ICAPS*. AAAI.

Piacentini, C.; Alimisis, V.; Fox, M.; and Long, D. 2013. Combining a temporal planner with an external solver for the power balancing problem in an electricity network. In *ICAPS*.

Roozemond, D. A. 2001. Using intelligent agents for pro-active, real-time urban intersection control. *European Journal of Operational Research* 131(2):293–301.

Shin, J., and Davis, E. 2005. Processes and Continuous Change in a SAT-based Planner. *Art. Int. (AIJ)* 166:194–253.

Veselý, V.; Rosinov, D.; and Foltin, M. 2010. Robust model predictive control design with input constraints. *ISA Transactions* 49(1):114–120.

Xie, X.-F.; Smith, S.; and Barlow, G. 2012. Schedule-driven coordination for real-time traffic network control.

Planning for Social Interaction with Sensor Uncertainty

Mary Ellen Foster
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
M.E.Foster@hw.ac.uk

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Abstract

A robot coexisting with humans must not only be able to perform physical tasks, but must also be able to interact with humans in a socially appropriate manner. In this paper, we describe an extension of prior work on planning for task-based social interaction using a robot that must interact with multiple human agents in a simple bartending domain. We describe how the initial state representation developed for this robot has been extended to handle the full range of uncertainty resulting from the input sensors, and outline how the planner will use the resulting uncertainty in the state during plan generation.

Introduction

A crucial aspect in the design of an interactive system is *state management*: transforming the noisy, continuous hypotheses produced by the low-level input processing components into a form that can be used as the basis for higher-level action selection by a component such as a planner. Intuitively, states represent a point of intersection between low-level sensor data and the high-level structures used for action selection. Since states are induced from the mapping of sensor observations to property values, the challenge of building an effective state manager rests on defining appropriate mapping functions. A state representation that considers only the highest-confidence inputs is straightforward to maintain and reason with, but discards a great deal of potentially useful information. On the other hand, a representation that takes into account the full set of input possibilities—along with their estimated confidence scores—can be more robust and informative, but requires more sophisticated methods of maintenance and more complex forms of reasoning and planning.

The particular application we consider here is a robot bartender called JAMES (Figure 1), which has the goal of supporting socially appropriate multi-party interaction in a bartending scenario.[1] In particular, the robot's sensors monitor two primary input modalities: vision and speech. Based on observations about the agents in the bar provided by these sensors, the system maintains a model of the social context, and decides on effective and socially appropriate responses in that context. Key to our approach is the use of a high-level planner for action selection in the robot system, in the place of a traditional interaction manager (Larsson and Traum 2000). Specifically, we use the knowledge-level planner PKS (Petrick and Bacchus 2002; 2004), a choice that is motivated by PKS's ability to work with incomplete information and sensing actions, since the robot will often have to gather information from its environment (e.g., by asking a customer for a drink order) in addition to performing physical tasks such as handing over drinks.

[1] See www.james-project.eu for more information.

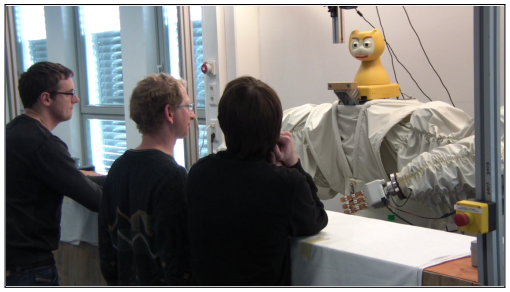


Figure 1: The JAMES robot bartender

In this paper, we describe how the initial, deterministic state representation has been extended to incorporate the full data from the robot's input sensors, and how the planner is using this enhanced representation during plan generation.

State Management with Uncertain Input

The task of the *state manager* in the robot bartender system is to keep track of information about the agents in the scene: for example, their locations, whether they are currently seeking the bartender's attention, and their drink orders. The state is derived from the continuous stream of messages produced by the low-level input and output components. In addition to storing low-level sensor information, we also infer additional relations not directly reported by the sensors; for example, we fuse vision and speech to determine which user should be assigned a recognised speech hypothesis, and use the vision data to estimate each customer's attention-seeking state (Foster, Gaschler, and Giuliani 2013).

Since the input provided by the vision and speech processing components is uncertain, there is an inherent uncertainty about the state. However, for simplicity, the state representation used in the initial JAMES system (Petrick and Foster 2013) stored only the highest-probability hypotheses, with no

seeksAttention(A1)	true	0.75
seeksAttention(A2)	false	0.45
lastSpeaker()	A1	1.0
lastEvent()	userSpeech(A1)	1.0
drinkOrder(A1)	green lemonade	0.677
	blue lemonade	0.322
lastAct(A1)	greet	0.25

Table 1: State excerpt, showing both the old discrete representation (highlighted portion) and the new representation

action ask-drink(?a : agent)
preconds: K(inTrans = ?a) $\wedge$ $\neg$K(ordered(?a)) $\wedge$ $\neg$K(otherAttnReq) $\wedge$ $\neg$K(badASR(?a))
effects: add(K$_f$, ordered(?a)), add(K$_v$, drinkOrder(?a))

Figure 2: Example PKS action in the bartender domain

confidence measures; a sample state using this representation is shown in the highlighted portion of Table 1.

This initial representation simplified action selection considerably, but also discarded potentially relevant information from the input sensors. We have therefore extended the initial version of the state manager to associate each hypothesis with a confidence score, and to include alternative hypotheses about a customer's drink order (Foster, Keizer, and Lemon 2014). Table 1 shows a full state using this expanded representation: in addition to the old-style state information in the highlighted portion, this state adds confidence scores to all properties—meaning that low-confidence relations like lastAct(A1) can now be included—and also includes multiple values for relations like drinkOrder(A1). The resulting representation is similar to the *Discrete* distribution used in RDDL (Sanner 2011), the language for the recent probabilistic tracks of the International Planning Competition.

Planning under Sensor Uncertainty

To generate plans for the robot, PKS uses a knowledge-level domain model that includes a specification of the physical, sensory, and linguistic (speech) actions available to it. The current domain supports simple interactions with individual agents for ordering drinks from the robot, as well as socially motivated behaviour such as group ordering and multi-party turn-taking. For example, Figure 2 shows the PKS representation for the ask-drink(?a) action ("ask an agent ?a for a drink order"), which is modelled as a sensing action that returns a placeholder (the function drinkOrder) for information that will become known at execution time.

We are currently improving our ability to plan with sensor uncertainty in the states described above. Since PKS does not (currently) work directly with probabilistic representations, we are modelling disjunctive state information like drinkOrder in Table 1 using PKS's ability to use "exclusive or" formula of the form $(\phi_1|\phi_2|\ldots|\phi_n)$ (which is interpreted as "one, and only one, of the ϕ_is is true"), ordered by decreasing confidence values. To incorporate confidence information for single-value relations, such as seeksAttention, we instead employ empirically determined confidence thresholds to determine whether to accept the current state information or to make an effort to gather more information before continuing. Updated state information is regularly sent to the planner from the state manager after action execution, and used for monitoring and replanning purposes.

Once the extended state information is available in the planner's knowledge state, it can be directly used during plan construction. In practice, such knowledge often has the effect of introducing additional sensing actions into a plan, to disambiguate between disjunctive alternatives. To aid this process, we are adding new actions which correspond to information-gathering (clarification) questions that the robot can ask to help clarify uncertain beliefs—without asking an agent to simply repeat an utterance, which is often interpreted by humans as a poor dialogue move (Skantze 2005).

Future Work

We will shortly carry out a user study to assess the impact of the new state representation and updated planning approach on user interactions with the system, comparing a version of the bartender that deals with all of the above forms of uncertainty to one that does not. Based on the behaviour of previous versions of the system—which did not incorporate state uncertainty but still performed reasonably well—we expect to see a positive impact in task performance (i.e., the number of drinks correctly served), since the bartender should clarify lower-confidence or ambiguous state hypotheses instead of simply serving what it believes to be the requested drink. On the other hand, it may be that the subjective user judgements will be negatively affected if the system clarifies too frequently in contexts where the top hypothesis is correct.

Acknowledgements

This research has received funding from the European Union's 7th Framework Programme under grant No. 270435.

References

Foster, M. E.; Gaschler, A.; and Giuliani, M. 2013. How can I help you? Comparing engagement classification strategies for a robot bartender. In *Proceedings of ICMI 2013*.

Foster, M. E.; Keizer, S.; and Lemon, O. 2014. Towards action selection under uncertainty for a socially aware robot bartender. In *Proceedings of HRI 2014*.

Larsson, S., and Traum, D. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering* 6(3–4):323–340.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of AIPS 2002*, 212–221.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of ICAPS 2004*, 2–11.

Petrick, R. P. A., and Foster, M. E. 2013. Planning for social interaction in a robot bartender domain. In *Proceedings of ICAPS 2013, Novel Applications Track*.

Sanner, S. 2011. Relational Dynamic Influence Diagram Language (RDDL): Language description.

Skantze, G. 2005. Exploring human error recovery strategies: Implications for spoken dialogue systems. *Speech Communication* 45(3):325–341.

Exploring High Dimensional Metric Spaces: A Case Study Using Hubble Space Telescope Long Range Planning

Mark E. Giuliano
Space Telescope Science Institute, 3700 San Martin Dr., Baltimore, MD 21217, USA
giuliano@stsci.edu

Abstract

Planning and scheduling applications often have multiple criteria or metrics that need to be satisfied and or optimized to create a plan suitable for execution. Applications also typically have corresponding program input parameters that impact the objectives. A multi-objective approach provides a framework for evaluating plans and the impacts of control parameters in terms of multiple criteria without having to combine criteria evaluations with a single objective function. However, the framework requires tools that allow end users to analyze alternate plans in terms of a Pareto-surface of competing solutions. This paper explores the utility of such tools through a case study exploring a high dimension metric space (32 separate metrics) for operational Hubble Space Telescope planning and scheduling data.

Introduction

Although the SPIKE system has been used for long range planning Hubble Space Telescope (HST) observations for over 20 years, the core requirements that determine the quality of a plan have only recently been formally specified. [Giuliano 2013] describes a set of metrics that were incorporated into the SPIKE system and outlines a study to validate the metrics using operational HST data. This paper presents the results of that study and provides a case study on the effectiveness of multi-objective graphical based tools for analyzing multi-objective solutions when exploring high dimension metrics spaces. The HST case study involves 32 separate metrics collected for 32 plans that varied seven SPIKE control parameters. The long term goal of this work is to streamline the long range planning process for HST and for the future James Webb Space telescope mission.

HST Planning and Scheduling

Launched in 1990, the Hubble Space Telescope is a general purpose space observatory that provides support for near-infrared, visible, and ultraviolet frequencies. HST has multiple science detectors that were designed to be upgraded and replaced during its mission. HST is in a low earth orbit approximately ~600 km above the Earth and orbits the Earth every 96 minutes. The main physical constraint on HST observations is that targets selected by the observer cannot be occulted by the Earth, the Sun, or the Moon. In addition, a user can place other requirements on an observation including the ability to specify time windows for observations (e.g. schedule OBS1 day 330-360), to link observations via precedence or grouping relationships with offsets (e.g OBS1 after OBS2 by 10-15 days), and to link observations via roll constraints (e.g. Same roll OBS1 as OBS2). The time intervals that satisfy all constraints are called observation *constraint windows*.

Hubble observations are carried out in a repeated yearly cycle. In each cycle, astronomers submit proposals for using the telescope to the Space Telescope Science Institute (STScI). The submitted proposals are ranked by scientific merit by an external Time Allocation Committee. Approved proposals are prepared for execution using the Astronomer's Proposal Tool (APT) and then submitted to STScI. Accepting a proposal represents a commitment by STScI to execute the observations to completion. There are no priorities within the pool of accepted proposals.

HST scheduling is handled in a two-phase process by separate long-range planning and short-term scheduling systems [Giuliano 1998]. In the first phase, long-range planning assigns observations to overlapping least commitment plan windows that are nominally 56 days long. Plan windows are a subset of an observation's constraint windows and represent a best effort commitment to schedule within the window. In the second phase, plan windows are used to create successive short-term schedules for 7-day upload periods. This two phase process allows a separation of concerns in the scheduling process: plan windows globally balance resources, are stable with respect to schedule changes, and provide observers with a time window so they can plan their data

reduction activities. The short-term scheduler provides efficient fine-grained schedules to the telescope.

At the start of each yearly cycle an ingest process, plans all observations from the current new cycle, on top of observations from previous cycles that have not already been executed. This paper presents a case study of the cycle ingest process based on the most recent HST observing cycle.

HST long range planning is implemented by the SPIKE planning and scheduling system. SPIKE (Johnston and Miller, 1994) is a constraint based planning and scheduling tool kit that was created for use on the Hubble Space Telescope. SPIKE models astronomical scheduling constraints and provides a rich set of search mechanisms which can be configured to explore the high value portion of the long range plan space (Giuliano 2013, Giuliano 2008).

The HST Cycle Ingest Process

The most recent HST long range planning cycle ingest took place in a three week period during the late summer of 2013. During this time operations staff used SPIKE to generate multiple candidate long range plans by running SPIKE multiple times with different runtime control parameters. The users explored the SPIKE input parameter space by performing 32 runs that varied seven SPIKE Boolean major control parameters (these controls are explained in a subsequent section). These runs explored the high value portion of the 128 possible control parameter assignments. Operations staff evaluated these runs using plan metrics and manual inspection of SPIKE graphics for resources and plan window placement. Based on this inspection an additional set of runs were performed using the best setting of the seven parameters and exploring additional spike parameters. Finally, all the schedules generated were considered and a single long range plan was selected for operations and released to the astronomical community so they could plan for data analysis. It must be stressed that the process of selecting a long range plan for operational release is currently highly intuitive involving the competing intuitions of three operations staff members.

The plan metrics and SPIKE runtime images for each of the runs were stored on disc and made available to the SPIKE development team. These metrics include not only metrics calculated by SPIKE itself but also metrics derived from SPIKE reports and plans based on tools written by the operations staff. The data was used as a basis for a study examining the metrics data in the winter of 2014. The goals of this exercise were:

- To validate the utility of graphical based tool in exploring and analyzing HST long range planning metric data
- To validate the choice of SPIKE control parameter by determining the impacts of SPIKE control parameters on the quality metrics.
- To validate the quality of the metrics by determining the extent to which metrics that nominally evaluate the same features correlate.
- To determine the extent to which metrics coded directly within SPIKE match metrics determined off line after the SPIKE processing has finished.

The overall purpose of the exercise is to feedback the findings into the next cycle ingest process.

HST SPIKE Plan Metrics

Figure 1 presents 32 plan metrics considered in this study. These metrics which evaluate the quality of a plan as a whole are classified into 3 broad categories and are marked as to whether or not they are internal to SPIKE or derived from SPIKE output products by user tools.

Of special concern to long range planning is the handling of orbital resources. There are approximately fifteen 96 minute orbits in a HST day. About nine out of the fifteen orbits in each day cross the South Atlantic Anomaly (SAA) a region off the coast of South America that has unusually high radiation. In each of these orbits, the SAA passage occurs in a slightly different portion of the orbit. These orbits are called SAA impacted orbits. No observations can occur during an SAA passage. However, we can schedule observations in SAA impacted orbits, if the Earth occultation for the observation occurs during the SAA passage. This case is called SAA hiding. Orbits without any SAA crossing are called SAA free orbits. In the most recent HST observing cycle 60% of the approved observations can be scheduled only in SAA-free orbits, whereas only 33.3% of the orbits are SAA-free. For any given target, SAA hiding occurs only for a small fraction of a year. In addition the HST orbit follows a 56-day north to south precession cycle. During the week where HST is in it's most northern part of the precession cycle it is especially hard to find observations that are schedulable as sun exclusion blocks northern targets.

Category	Metrics	Purpose
Resources	*global-resource* *orbit-criteria* *saa-orbit* *orbit-distribution* *orbit-pcf-usage* res-report res-report-over res-report-over-free north-point1 north-point2 saa-another res-report-v res-report-over-v res-report-over-free-v north-point1-v north-point2-v saa-another-v	To assign plan windows that balance resources across the planning session. Especially resources involving the SAA and the north point SAA. Criteria in italics are internal to SPIKE while not italic criteria are from SPIKE outputs using operational scripts. Spike has a mode in which it can validate its own least commitment resource model by assigning observations to precise times. Metrics with a –v are from those runs while the non –v reports use a least commitment resource model. All metrics are translated into minimization criteria within the SPIKE model.
Window Size	*window-size-plan* mean-pw-dur percent-1 percent-2 percent-3 percent-26 percent-41 mean-pw-dur-sr percent-1-sr percent-2-sr percent-3-sr percent-26-sr percent-41-sr	To assign plan windows that come close to the nominal size of 56 days and to minimize to many short plan windows. The percent metrics 1,2,3, minimize the number of days that have 1,2,3 .. day plan windows overlapping them. The percent metrics 26, and 41 are maximization criteria. The metrics with –sr limit the results with respect to special requirements. For example, an astronomer can put a requirement that limits scheduling to a week. The –sr metrics take this into account while the non –sr metrics do not.
Packing observing programs	*pack-proposal* clustering clustering-v	To plan visits from a single observing program as close a possible in time. This helps STScI to complete programs.

Figure 1: SPIKE Plan Metrics

Graphical Tools for exploring Pareto Surfaces

A major goal of the study was to examine the utility of tools for exploring Pareto-surfaces to select a solution for execution. A first result is that with 32 metrics 31 out of the 32 schedules generated are not strictly dominated by another. With a high number of metrics, just considering non dominated solutions does not winnow the search space effectively. This result held even when subsets of metrics were considered. A second result is that tools such as those described in [Giuliano Johnston 2010, Giuliano Jonahston 2011] get overwhelmed by the high number of dimensions. Even though these tools are dynamic they are not flexible enough to provide users with the visualization tools needed to explore a complex space. As part of the case study a new set of tools were developed to address these problems. The tool is based on the d3 java script library and has the following features:

- Pareto surfaces and metrics are represented in the SPIKE domain model;
- Within the domain model end users have the ability to dynamically create alternate Pareto-surfaces by selecting the metrics under consideration, and the candidate set of potential solutions to consider;
- Users can create charts that get displayed in browser windows. The user can select the type of data displayed (e.g. raw criteria scores, rank index of scores, normalized criteria values).
- The user can also control color annotations in making a chart.
- The charts themselves allow features such as reordering axis and brushing indexes.

Figure 2a shows a chart considering a Pareto surface made from the initial 32 runs and 16 metrics that evaluate the schedules in terms of resources and SAA hiding. The chart is color coded so that solutions with a specified set of

control parameters (see the next section) are in green, the solution selected for execution is in blue, and the rest are presented as red.

Even with a smaller set of 16 metrics, that only measure resource related issues, the parallel coordinate chart is very messy and it is hard to see patterns. Users utilize some metrics as strict criteria and others as diagnostic information that indicates the need for manual inspection of the plan for issues. As such users need the ability to flexibly define the metrics and metric ranges that are considered in making a Pareto-surface. Figure 2b shows the same chart where the chart has been brushed to select a specified range of values for a given criteria.

The new graphical tool set is both flexible and light weight reusing browser capabilities. All charts presented in this paper are available in the tool set. However I have substituted hand produced charts in some cases from Excel as they currently look better. The plan for the next HST cycle ingest is to further develop these tools and to provide them to end users for use in the operational ingest process.

The impact of SPIKE Control Mechanisms

The SPIKE long range planning algorithm is driven by three software mechanisms called criteria, critics, and filters. By default SPIKE plans an observation by iterating through the schedulable start times for a plan window at the granularity of a day. For each day a potential plan window is generated consisting of any suitable time over the next 56 days (56 days being the nominal plan window size). The resulting window is evaluated by criteria and the best window according to the criteria evaluation is selected. This process is modified by critics and filters. Critics sculpt the 56 day window by ensuring that certain properties apply. For example, an SAA critic cuts plan windows that contain SAA regions to end and start at week boundaries. This mechanism ensures we explore the high value portion of the search space. Filters prevent SPIKE from iterating over the entire planning horizon. SPIKE has a series of filters each of which define a subset of the planning horizon for a visit. The filters are ordered from most restrictive (i.e. highest value) to least restrictive and SPIKE attempts to use each filter in turn ensuring that SPIKE will assign a window with the most restrictive filter (i.e. the highest value feasible portion of the search space).

Figure 3 gives a taxonomy of SPIKE search mechanisms. It shows that window criteria, critics, and filters directly guide the selection of windows by SPIKE while plan metrics currently only perform post plan evaluations. A long term goal of this work is to move the plan metrics into a guiding roll within the SPIKE process.

Mechanism	What they do	Impact
Window Criteria	Evaluates potential windows for observations	Guide the plan windows selected by SPIKE
Critics	Sculpts windows	
Filters	Reduces the search space	
Plan Metrics	Evaluates plans	Evaluate Entire plans do not currently guide plan creation in SPIKE

Figure 3. A Taxonomy of SPIKE Control Mechanisms

The SPIKE Pareto surface mechanism was augmented to measure the impacts of SPIKE control parameters on the quality of solutions. While SPIKE has many types of criteria, critics, and filters the mechanisms varied in the cycle ingest process all involve resources and the SAA. The operations staff varied whether the following control mechanisms were used in the runs:

- *Great-attractor* (GA). This new criteria determines the possible demand for each orbit in the session and prefers to plan observations in times with the least amount of demand.
- *North point* and *SAA criterion* – When possible prefer to plan visits in times where they have north point SAA or SAA hiding.
- *North point* and *SAA critics* – Sculpt plan windows to only include weeks which have good north point SAA or SAA hiding
- *North point* and *SAA filters* – If an viable solution can be found with north point SAA, or SAA hiding do not consider other time intervals.

These parameters define a set of seven Boolean controls. Operations staff examined 32 different settings of these parameters. Figure 4a shows how the values for 16 resource related metrics change in aggregate when comparing runs with and without the particular control feature turned on. Here a positive number means the turning on the control feature improved schedules with respect to a plan metric and a negative number means the control feature made the plan metric worse. As can be seen the great attractor is strongly associated with improving plan metrics. This criteria was new in this cycle and differs from the others in that it is based on the actual resources available to schedule in a cycle. We know from experience that SAA orbits are hard to fill but it is still the case that some are harder to fill than others based on the distribution of targets in the given cycle.

The controls in italics are those that the operations staff chose as the best settings for further exploration. In general these controls have the most positive impacts on all of the resource criteria and no or minimal negative

impacts. This chart strongly validates the choice of control settings by the operations staff.

Figure 4b shows the same control chart but this time plots the impacts on plan metrics that evaluate an LRP in terms of preferring that windows for a single science program are close in time and maximizing plan window size. Here the results show that turning on the controls generally makes these metrics worse (i.e. most impacts are below zero). This is logical as these controls prefer other features than large windows and packed programs. In general the selected controls perform least poorly (with the exception of the SAA filter). These results suggest that SPIKE should have corresponding control mechanisms that in effect lobby for these plan metrics.

Validating SPIKE Metrics

In this part of the study we examined the internal consistency of the SPIKE metrics. Do metrics that nominally measure the same feature typically have the same score? Are there metrics that can be removed as redundant? Do the metrics that are internally part of SPIKE correlate with the metrics produced externally from SPIKE by user tools. This last question is important as the goal is to move metrics into SPIKE so they can be used to guide the generation of plans.

The ability to reason about metric correlations was added to the multi-objective model and graphical displays were defined. Figures 5a-c present bar graphs showing the correlation between the groups of each type of metric. The chart shows the average variation in criteria values between each pair of criteria in the group. The variation is measured for metric scores normalized to be between 0 and 1. The greater the value the smaller the correlation.

The results are mixed. Figure 5a shows that each of the three proposal packing metrics maps within 15% of the other metrics. The internal SPIKE proposal packing metric is sufficient. The window size plot (Figure 5b) shows a very strong correlation between the metrics with and without special requirements. These vary only by 1-5%. In future cycles both sets of metrics do not need to be considered. The internal SPIKE plan window size metric matches within 8% of the external mean pw duration metric. The outlier is the 26 day percentage metric which does not correlate with the other metrics. A better understanding of this metric needs to be developed.

The correlation for the resource related metrics is relatively poor with few metrics correlating at 15% and most correlating around 25% or higher. In hindsight this is not unexpected. Quantifying resource usage is the most complex part of the HST long range planning process and SPIKE has three different models of resource consumption used in planning [Giuliano et al 2013]. The fact that multiple measures have competing results shows the different perspectives taken by the models and the complexity in modeling least commitment windows.

Conclusions

The results of a plan metric evaluation study using data from the most recent HST observing cycle were presented. The study examined data for 32 plan metrics on 32 scheduling runs which varied 7 SPIKE control parameters. The study showed the utility of highly dynamic tools to explore Pareto surfaces. Users need the ability to define the metrics of interest and to produce charts that can be graphically manipulated. The study validated the choice of control parameters selected by SPIKE operations. Finally the study examined how groups of metrics correlate with each other. The goal of the study is to provide feedback into the operations process so we can improve the long range planning process. The long term goal is to move the plan evaluation process form its current intuitive based state to one based on graphical based presentations of validated metrics.

Acknowledgements

Thanks to Ian Jordan and Dave Adler for providing the cycle 21 operations data and for providing an operational perspective. Thanks to Andrew Myers for help with developing the d3 graphics code.

References

Johnston, M. and Miller, G. 1994. *Spike: Intelligent Scheduling of Hubble Space Telescope Observations*. In Zweben M. and Fox M. eds. *Intelligent Scheduling*, 391- 422. Morgan-Kaufmann.

Giuliano M.E. 2013. *The Mystery of the Missing Requirements: Optimization Metrics for SPIKE Long Range Planning.* International Workshop on Planning and Scheduling for Space (IWPSS) NASA Ames Research Center, California

Giuliano, M.E., and Johnston, M.D. 2011. Developer Tools for Evaluating Multi-Objective Algorithms. *International Workshop on Planning and Scheduling for Space,* Darmstadt, Germany.

Giuliano, M.E, Hawkins R, Rager R,. 2011. A Status Report on the Development of the JWST Long Range Planning System. *International Workshop on Planning and Scheduling for Space (IWPSS)*. Darmstadt, Germany.

Giuliano, M.E., and Johnston, M.D. 2010. Visualization Tools For Multi-Objective Algorithms. Demonstration: *International Conference on Automated Planning and Scheduling (ICAPS)*, Toronto, Canada.

Giuliano, M.E., 2008. Handling Oversubscribed Orbital Resources in Hubble Space Telescope Operations. *Workshop on Oversubscribed Planning and Scheduling, at the International Conference on Automated Planning and Scheduling (ICAPS)*, Sydney Australia.

Giuliano, M.E. 1998. Achieving Stable Observing Schedules in an Unstable World. In *Astronomical Data Analysis Software and Systems VII*. 271-274.

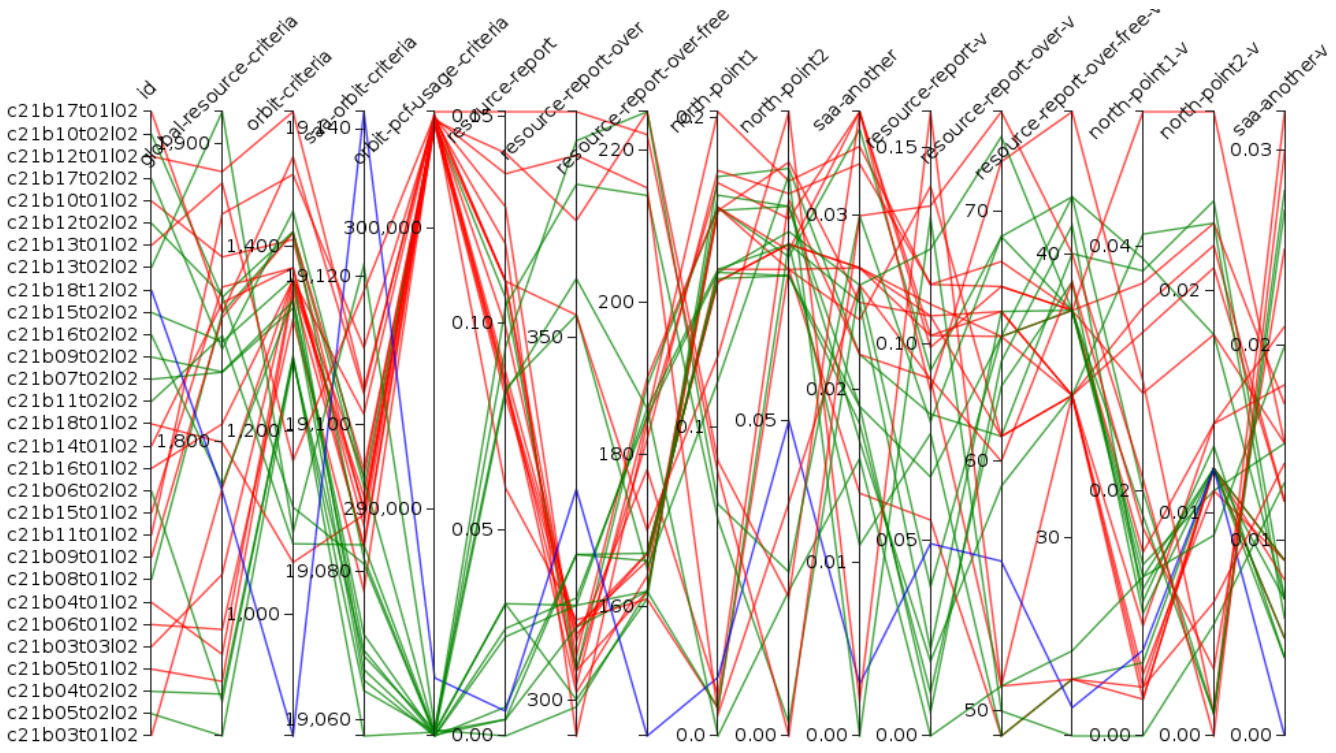
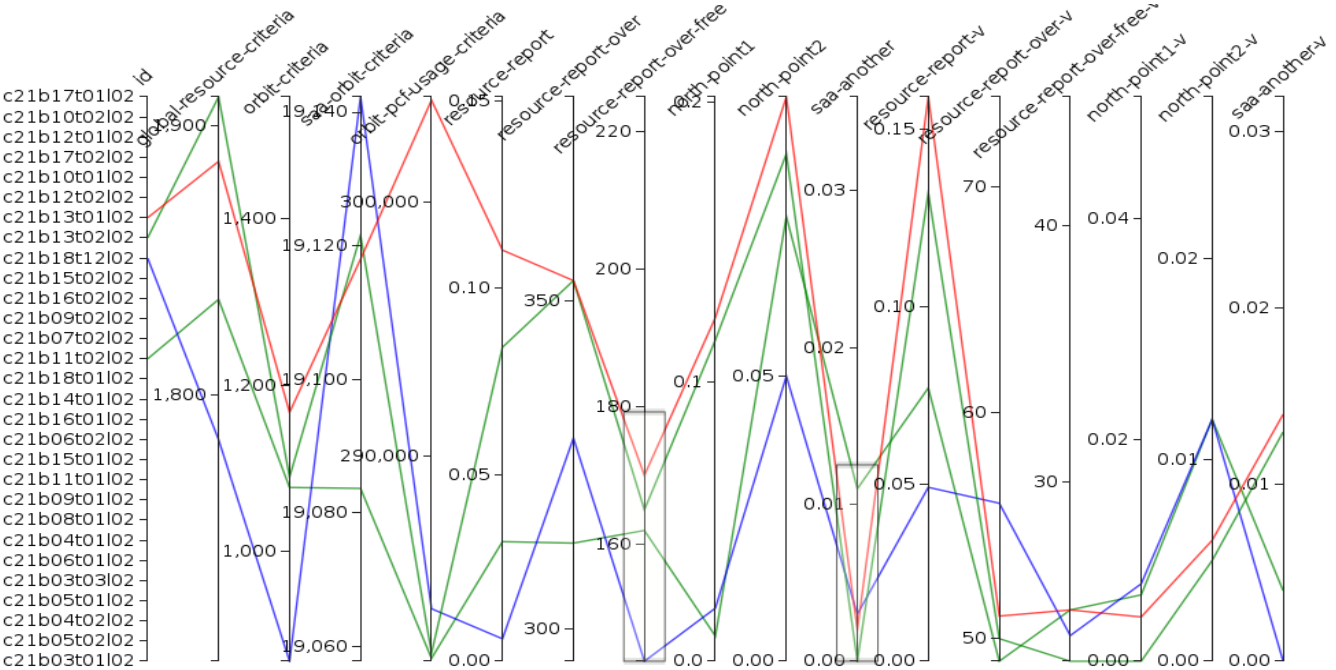


Figure 2. Parallel coordinate charts showing values for 16 resource related metrics. Each line represents a long range plan. Lines in green have a specified set of SPKE control parameters on. The line in blue is the solution selected for operations. Figure 2a (above) is the raw data of the chart. Figure 2b (below) is the chart with ranges selected (gray rectangles) for two metrics.



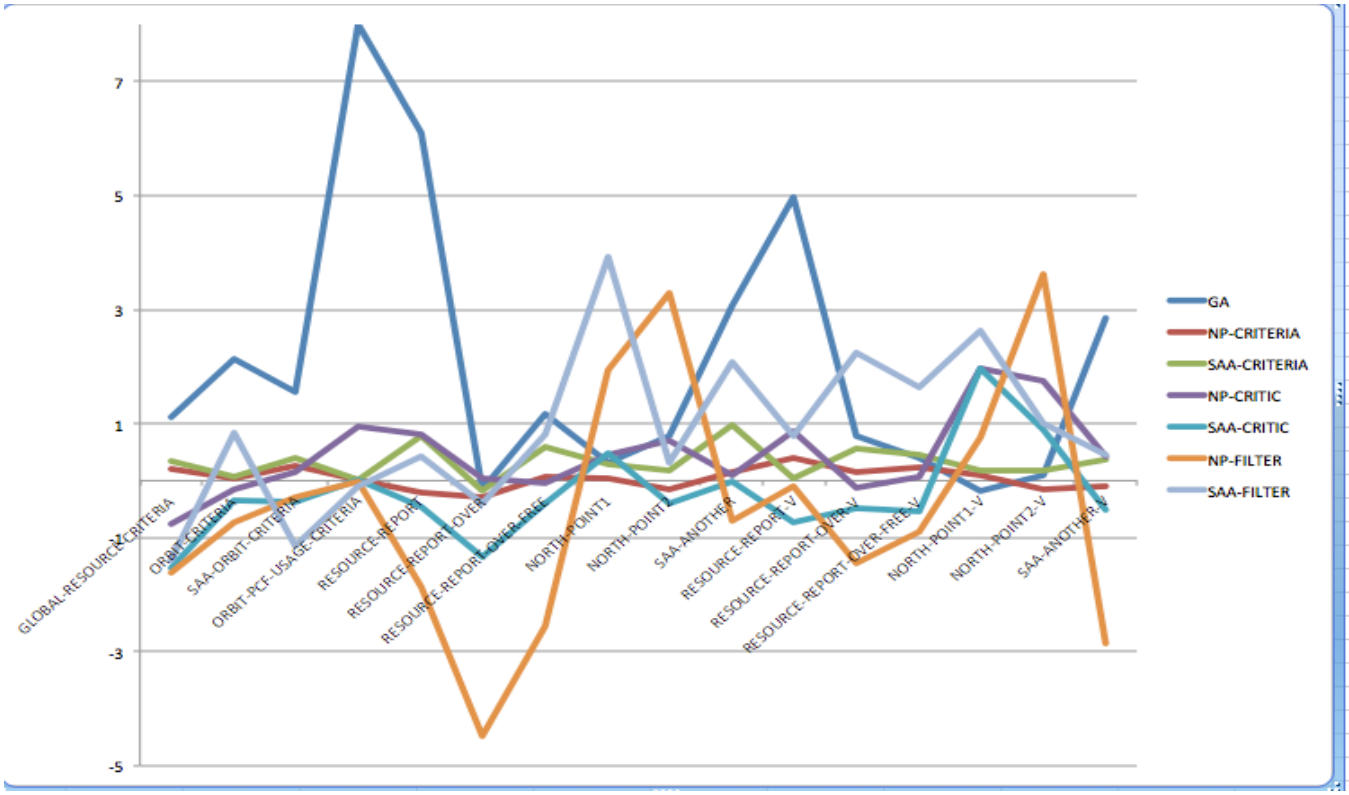
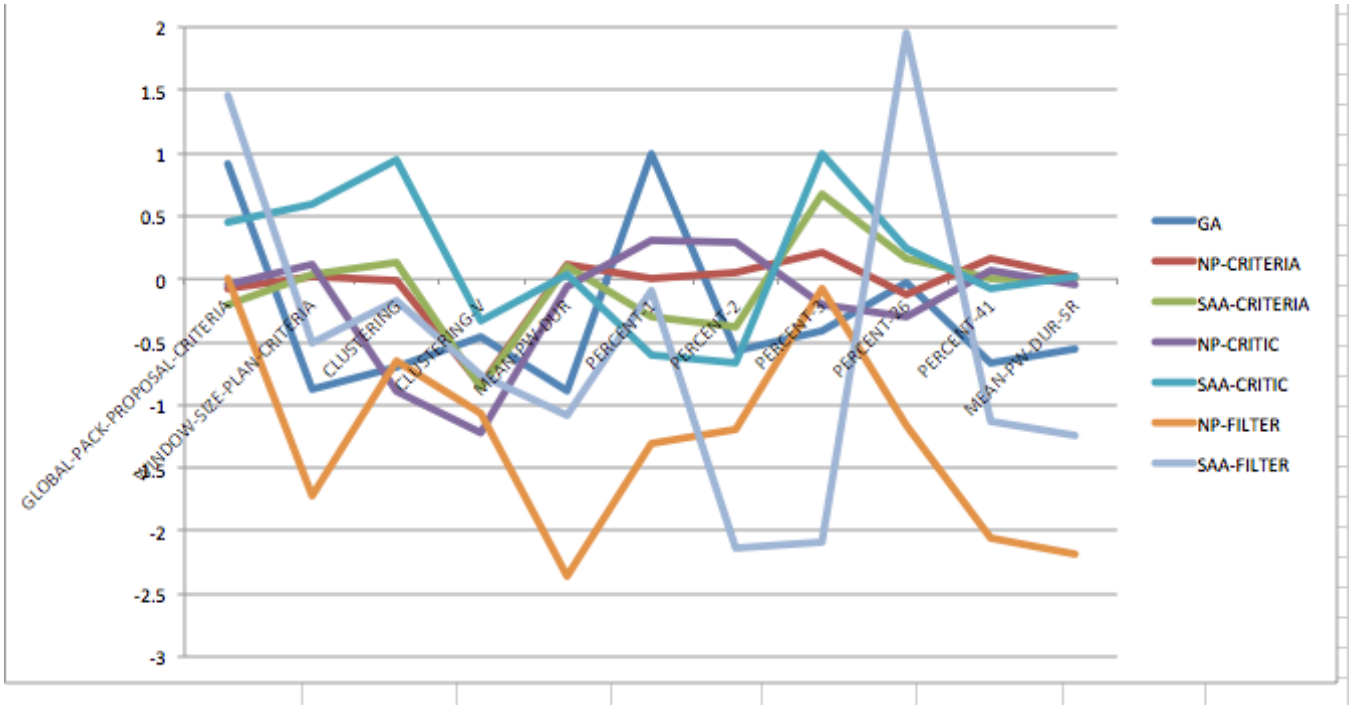


Figure 4. Parallel coordinate plots showing the incremental impact of turning on 7 different SPIKE control parameters on a specified set of resources. Each line represents turning on a SPIKE control feature. Values above zero represent a positive effect. Values below zero are a negative effect. Figure 4a (above) shows impacts for resource related metrics. Figure 4b (below) shows impacts for window size and program packing metrics.



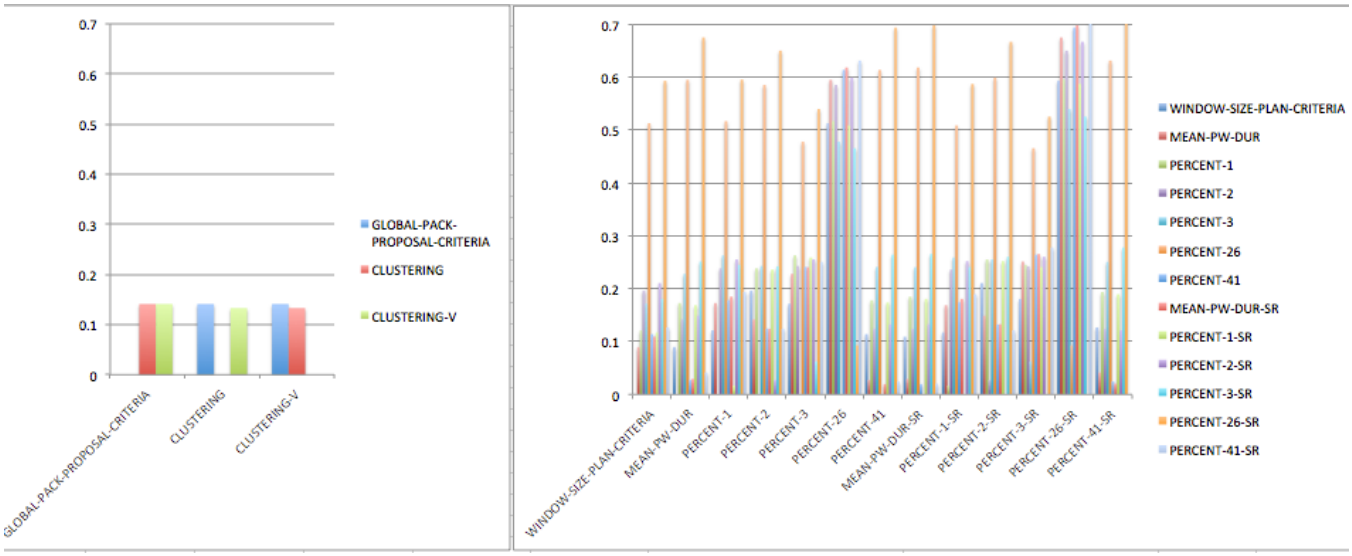
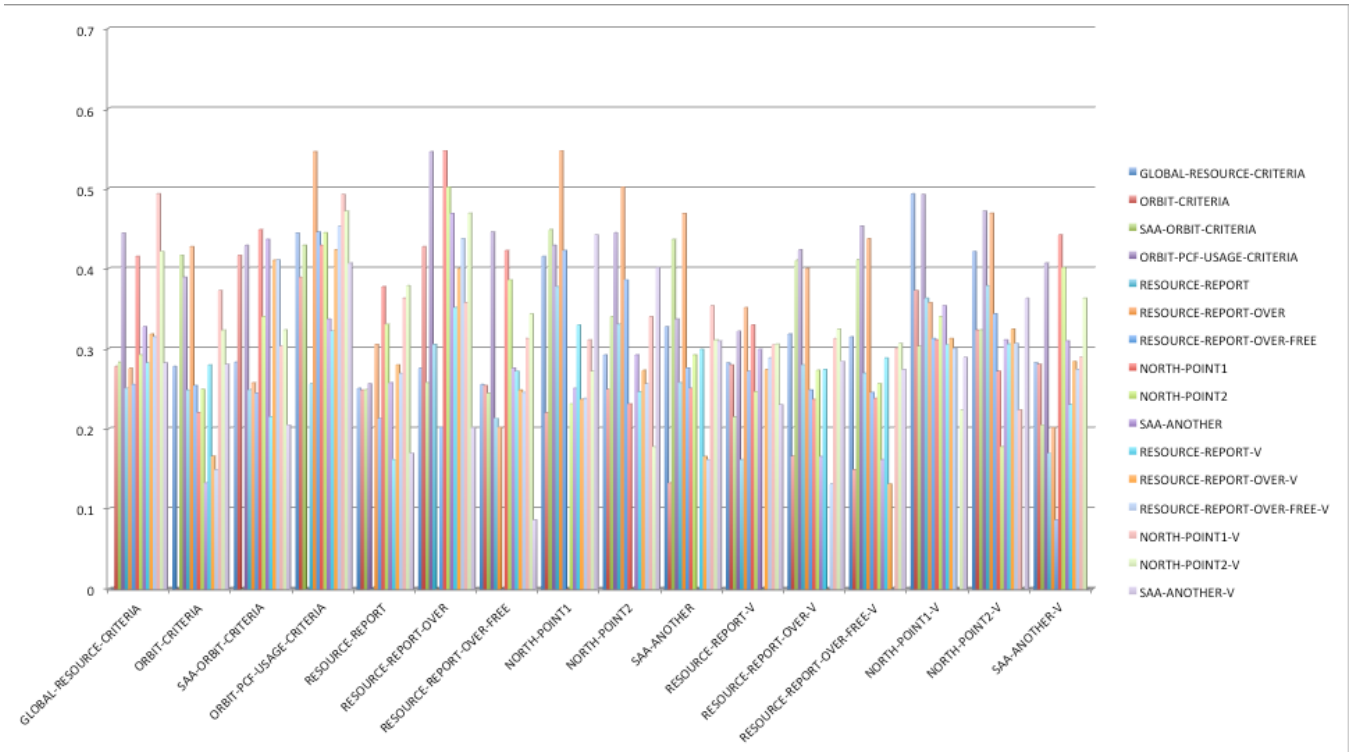


Figure 5. Bar graphs showing the correlation between pairs of metrics within a category (see figure 1 for categories). The value is the average difference in a 0-1 normalized score. Smaller values indicate better correlations. Note that the scales are different in each graph. Figure 5a (upper left) gives correlations for proposal packing criteria. Figure 5b (Upper right) gives correlations for window size criteria. Figure 5c (lower) gives correlations for resource criteria.



Intelligent UAS Sense-and-Avoid Utilizing Global Constraints

Javier Barreiro and **Minh Do** and **David E. Smith**
NASA Ames Research Center
Moffett Field, CA 94035
javier.barreiro, minh.do, david.smith@nasa.gov

Abstract

Sense-and-avoid (SAA) is a critical research topic for enabling the operation of Unmanned Aircraft Systems (UAS) in civilian airspace. SAA involves two planning related problems: 1) plan-recognition to predict the future trajectory of nearby aircraft, and 2) path planning to avoid conflicts with nearby aircraft that pose a threat. We have designed and built components of a novel intelligent sense-and-avoid (iSAA) reasoning framework that takes into account information about aircraft type, transponder code, communications, local routes, airports, airspace, terrain, and weather to more accurately predict near- and medium-term trajectories of nearby aircraft. By using this additional information both the onboard control software and the ground-based UAS operator can make more informed, intelligent decisions to effectively predict and avoid conflicts and maintain separation. While this capability benefits all categories of UASs operating under both Instrument Flight Rules (IFR) and Visual Flight Rules (VFR), it is absolutely essential for allowing smaller UASs to operate VFR at low altitude in uncontrolled airspace for operations such as survey work, wildlife tracking, aerial photography, utilities inspection, crop dusting, and package delivery.

1 Introduction

Unmanned Aircraft Systems (UAS) come in a wide range of sizes, from tiny to quite large as illustrated in Figure 1. As you would expect, they have an equally wide range of characteristics and capabilities, including cost, operating altitudes, range, speed, instrumentation, communication abilities, sensors, and payload capacity. To date, UASs have been used predominantly for military applications, but there is growing demand to employ these vehicles for a wide range of civilian purposes, including such things as: search and rescue, traffic monitoring and reporting, wildlife monitoring and surveys, fire and flood monitoring, pipeline and transmission line inspection, aerial photography, cropdusting, and package delivery. All of these applications, require operation of UASs in civilian airspace at lower altitudes. To date, UASs have essentially been operated under Instrument Flight Rules (IFR), which means that they have to adhere to a strict flight plan, and Air Traffic Controllers (ATC) have authority over their operation and route. This is not practical for smaller vehicles, for vehicles operating at low altitudes (below radar coverage), and for many of the applications envisioned for these vehicles. To operate in this environment,

Figure 1: Some small, medium, and large UASs with widely varying performance and capabilities.

these vehicles must be able to operate under Visual Flight Rules (VFR), and be able to Sense and Avoid (SAA) other aircraft in the same way that most small aircraft currently operate.

There are many aspects to the SAA problem including sensor technology, sensor integration, communication technology and security, threat detection, and threat resolution. Because of the interest in UASs, there has been a great deal of work on many of these topics. In this paper, we describe our preliminary work on threat detection and threat resolution. We assume a variety of inputs from various possible sensors and sources for use in threat detection. In the next section, we describe some related work on threat detection and threat resolution. We then motivate our approach in relation to this previous work. In Section 4 we lay out the architecture for our approach in greater detail, and describe our initial prototype implementation of threat detection utilizing a dynamic Bayesian Network. Section 5 describes current work on improving our prediction algorithm by automatically generating dynamic Bayesian Networks tailored to the location of the intruder. Section 6 describes future work on developing medium-term threat resolution software utilizing fast online probabilistic path planning.

2 Related Work

There has been a great deal of work sponsored and/or conducted by the FAA, NASA, the US military (e.g. GILA, JASMAD), and European agencies on automation of air traffic control and deconfliction of airspace. Most of this work focuses on recognition and resolution of conflicts at the Air Traffic Control (ATC) level between aircraft with known, well-defined routes. It assumes that at least one aircraft is in communication with ATC, and that the aircraft are "visible" to ATC (under radar coverage). This paradigm does not apply to smaller aircraft operating under Visual Flight Rules (VFR) at low altitudes – in many cases these aircraft are not in communication with ATC, may not be visible to ATC, and are not required to file or follow a flight plan. In these cases, the aircraft must sense and avoid each other without any ATC guidance.

There has also been considerable work on the SAA problem (Federal Aviation Administration 2009; Barnhart et al. 2012). Most relevant to this paper is work on TCAS (Federal Aviation Administration 2011), ACAS X (Kochenderfer, Holland, and Chryssanthacopoulos 2012), and JOCA (Chen et al. 2009).

Since 2000, the Traffic Collision and Avoidance System (TCAS II) (Federal Aviation Administration 2011) has been required in many countries (including the US, Europe, China, Australia and India) on all large commercial transport aircraft. TCAS is designed to reduce the chances of mid-air collision with other transponder or TCAS equipped aircraft by issuing advisories and threat resolutions to the TCAS equipped aircraft. It does this by 1) projecting the current track of any nearby aircraft (*intruder*) into the future, 2) recognizing if this poses a threat of a Near Mid-Air Collision (NMAC), and 3) issuing a Traffic Advisory (TA) if there is danger of an NMAC within 20-48 seconds, and 4) issuing a Resolution Advisory (RA) to avoid the NMAC if the danger

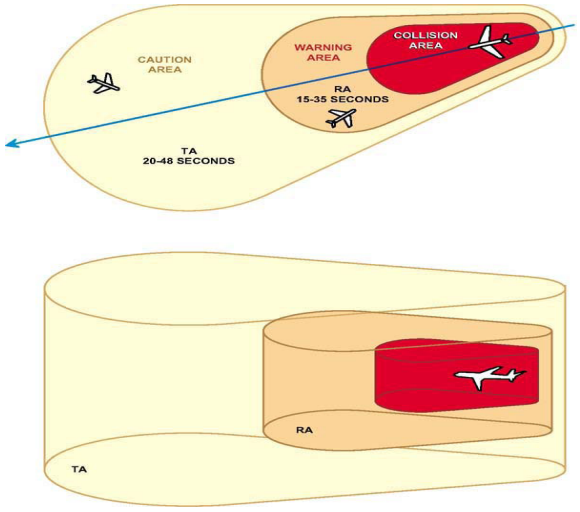


Figure 2: Scope of traffic advisories (TA) and resolution advisories (RA) for TCAS II.

is within 15-35 seconds. If both aircraft are TCAS equipped, the advisories are automatically coordinated to ensure that the responses do not conflict. Figure 2 illustrates the scope of TCAS reasoning and advisories.

While TCAS has reduced the incidence of NMACS for larger aircraft, there are a number of limitations with the system. First, the cost, hardware and power requirements limit installation to larger aircraft. Second, the system projects that the intruder will continue on its current course. While this may be a reasonable assumption for en-route aircraft operating at higher altitudes, it is less accurate, and less robust within the terminal area, particularly for smaller intruders operating VFR at low altitudes. Third, the resolution logic consists of hand generated heuristic rules, which are incomplete and difficult to verify. Finally, the system is designed for only short-term conflict resolution (15-48 seconds from NMAC). As a result, the Resolution Advisories are fairly aggressive maneuvers, and are limited to actions of climbing, descending, and otherwise constraining vertical speed. There are no horizontal avoidance actions. Note that vertical RAs could often be problematic for aircraft operating VFR at low altitude – descent may not be safe due to terrain, and climbing may not be possible due to clouds or airspace restrictions above.

The ACAS X system (Kochenderfer, Holland, and Chryssanthacopoulos 2012) is a proposed successor to TCAS II that is currently undergoing testing and evaluation. ACAS X improves on several of TCAS II's limitations – in particular, it is 1) capable of utilizing positional information from a broader range of sources, 2) it uses a probabilistic model of the intruders likely trajectory, and 3) the resolution logic is based on offline solution of an MDP, and is therefore more systematic, complete, and robust. As a result, ACAS X has demonstrated the ability to resolve more conflicts, while issuing fewer unnecessary resolution advisories. The architecture of ACAS X is shown in Figure 3. Unfortunately, ACAS

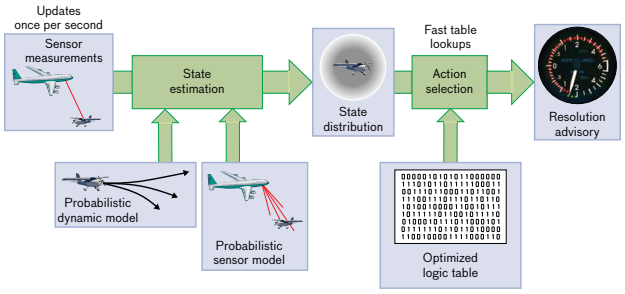


Figure 3: Architecture of ACAS X, from (Kochenderfer, Holland, and Chryssanthacopoulos 2012).

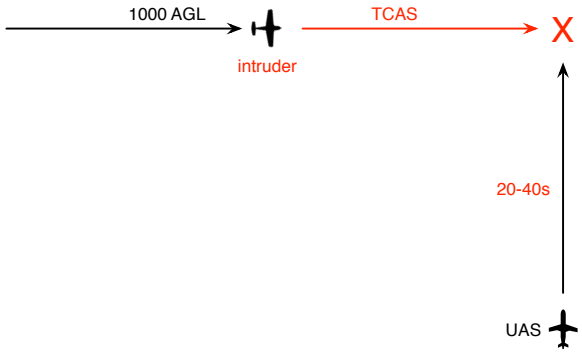


Figure 4: Intruder prediction for TCAS II.

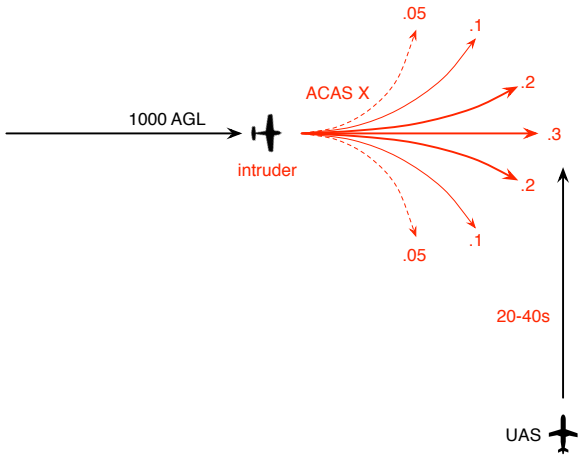


Figure 5: Probabilistic intruder prediction for ACAS X.

X is still aimed at short-term conflict resolution, and the advisories are still limited to climbing, descending, and constraint of vertical speed.

3 Approach

In this work, we are interested in addressing medium-term conflict avoidance for small UASs operating VFR at low altitude in a mixed air traffic environment. By medium-term we mean detecting and resolving conflicts between 30 and 120 seconds before they would occur. By doing this, we can maintain greater separation between aircraft, and conflicts can be resolved using less aggressive and less costly maneuvers. On the negative side, it is more difficult to accurately predict the trajectory of another aircraft this far in advance. The focus on VFR operation at low altitudes also presents challenges: 1) trajectories are less predictable for VFR traffic, and 2) terrain and weather may be a significant factor in predicting those trajectories and resolving conflicts – notably, aircraft operating VFR are required to maintain certain clearances from clouds and terrain, both vertically and horizontally.

We regard the problem of predicting the future trajectory of an aircraft as probabilistic plan recognition. Conflict resolution is a problem of path planning under uncertainty. We believe that both of these are knowledge intensive processes that rely on more than just the previously observed trajectory of the aircraft. In particular, prediction needs to take advantage of information about such things as aircraft type, local routes and traffic patterns, transponder operation, communications, terrain and weather. Conflict resolution needs to take account of many of the same things as well as the goals of the UAS.

To illustrate this, consider the simple example shown in Figure 4. Here, a UAS and another aircraft (the intruder) are both at 1000 ft AGL (above ground level) on intersecting courses. TCAS II would predict that the intruder would remain on its present (straight and level) course resulting in a conflict at the point X. Somewhere between 25 and 40 seconds before the intersection, the TCAS logic would advise an aggressive climb to avoid the intruder.

Figure 5 shows the difference for the ACAS X model: here there is a probabilistic model of the possible future positions of the intruder (derived from real data of aircraft tracks). As a consequence, the collision avoidance logic would recommend a maneuver to avoid the most time critical and likely paths for the intruder. In some cases, the logic will even delay making a decision until more evidence of the intruder's likely path is available.

In Figure 6 there is one piece of additional information available – the intruder aircraft is downwind from Runway 27 at an uncontrolled airport. In this case, if the aircraft is level or descending it is much more likely it is in the traffic pattern for this airport, and will be turning on a base leg, and then final leg as shown in Figure 6. As a result, the probability distribution for the future location of the aircraft should be strongly biased towards turning base. If we have additional information that the aircraft is descending, or that the landing gear is down, then it is even more likely that the aircraft will turn base and there will not be any conflict. In contrast, if we know that the aircraft is climbing, or that the landing gear is up, or that it has a discrete transponder code then it is more likely that the aircraft is on a downwind departure from the airport, and will not be turning base. Likewise, if the wind is strongly favoring the opposite runway (Runway 09), or the traffic pattern is on the south side of the airport rather than the north side (left traffic instead of right traffic), or the runway is too short for this type of aircraft,

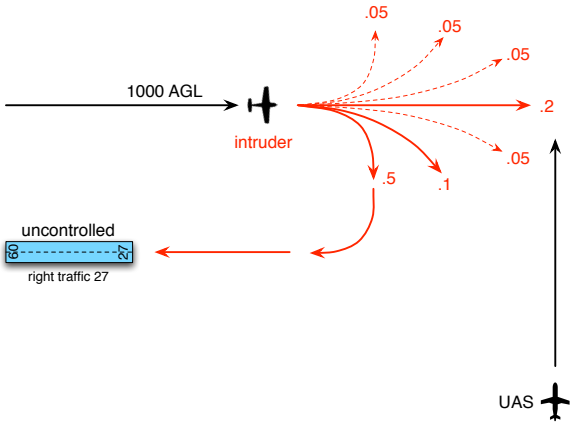


Figure 6: Intelligent intruder prediction.

then it is even less likely that the aircraft will turn base; in fact, it is unlikely that the aircraft is in the traffic pattern for this airport at all.

For these examples, the presence of an airport and the nature of its traffic pattern provided strong bias on the likely path of an intruder. However, there are many other situations that can provide such bias. For example, if an intruder has been observed performing abrupt changes in altitude and heading it is likely engaged in aerobatics. Type of airspace, weather conditions, time of day, the presence of an active aerobatic box, and aircraft type can provide additional evidence. Similarly, an intruder observed following a feature such as a coastline, a river valley, or a road is more likely to continue doing so instead of continuing on the current heading.

In all these cases, information about local routes, aircraft type, wind conditions, weather conditions, ceiling, topography, and other factors. All of these factors are evidence that needs to play a role in the prediction problem.

4 System Architecture

Sense-and-avoid procedures generally involve the following steps: 1) **intruder detection –** identify nearby aircraft, 2) **threat detection –** predict the trajectory of potential intruders and decide if any may cause a threat to the intended flight path, 3) **threat resolution –** devise an appropriate course of action to avoid the threat.

Figure 7 shows how these different components work together to provide the pilot/operator with a set of evasion maneuvers whenever a threat is detected. The Airspace Concept Evaluation System (ACES) (Airspace Systems Division, NASA Ames Research Center) is a simulation environment for the National Airspace that has been developed at NASA Ames Research Center. ACES includes an example SAA implementation called GenericSAA which follows this architecture. Since GenericSAA constitutes a working implementation that is compatible with our conceptual model of the problem, we have adopted it and enhanced it to fit our approach.

The **intruder detection** component is relatively straightforward, as it consists of applying specific vertical and horizontal separation thresholds for any aircraft detected in the vicinity of the ownship. In some cases any detected aircraft will automatically be considered an intruder, depending on the capabilities of the sensors available to the aircraft.

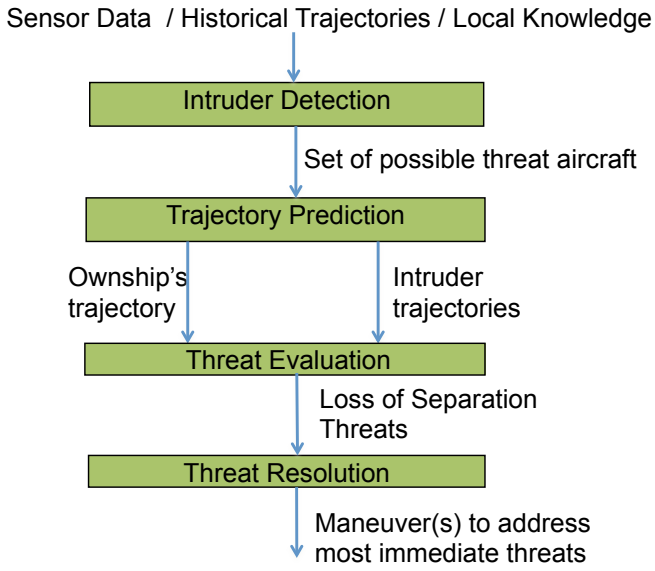


Figure 7: SAA Software Architecture

The **threat detection** component consists of **trajectory prediction** and **threat evaluation**. We have formulated a probabilistic approach that predicts the trajectories of potential intruders given their historical paths and relevant knowledge about the aircraft and the region. Figure 8 shows the components of our trajectory prediction framework, which feeds into the threat evaluation component:

Leg Extraction: unlike existing algorithms that utilize only the intruders observed trajectory as points to help predict short-term future trajectory points, we go one step further in extracting trajectory legs. A trajectory leg is a continuous portion of the trajectory where the aircraft is in a particular mode such as: climbing, turning right, flying straight, etc. A sequence of legs reveals the flights pattern and helps predict subsequent legs. We have built new software to extract legs from the observed state information for an aircraft based on three state variables: 1) speed; 2) altitude; and 3) heading.

Route Database: To understand the intent of each trajectory leg, we match it against defined route segments in the area. For example, if an observed aircraft is downwind for an active runway, is at traffic pattern altitude, and is level or descending, its very likely that it will subsequently turn on a base leg for the runway. We constructed a database of legs for traffic patterns, instrument approach procedures, and airways for an interesting test area near Sacramento, CA that contains several airports and a mix of different kinds of traffic. Each leg is described by its name, type, start and end fixes, altitudes for those fixes, heading, lateral tolerance, al-

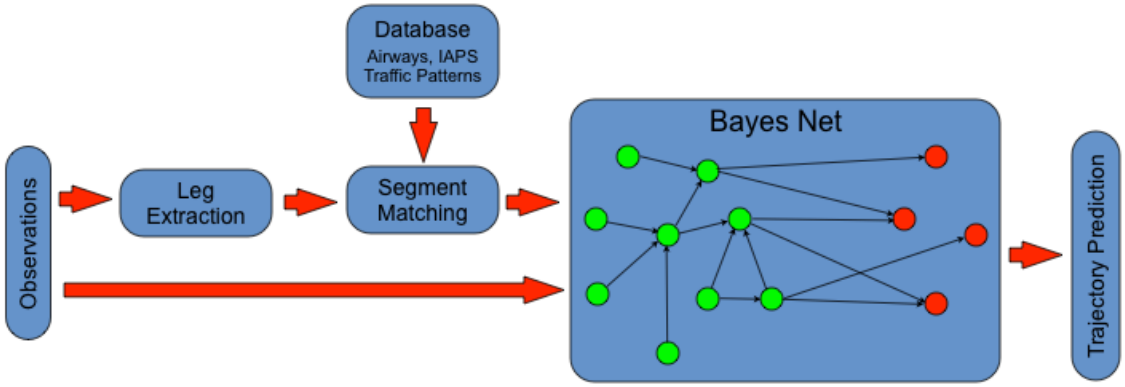


Figure 8: Trajectory Prediction Architecture

titude tolerance, and heading tolerance. These tolerances are much tighter for something like an instrument approach than for something like a federal airway, or crosswind departure.

Segment Matching: we developed and built an algorithm to match the trajectory legs, created from the observed history of an intruders trajectory, to the segments in the route database. Our matching algorithm decides that a leg L could be a given route segment S in our database if all points in L 1) are located within the geometric region defined by Ss start/end points and tolerances, and 2) have heading bounded by Ss heading and heading-tolerance values.

Predicting Future Intruder Trajectory: we use a Dynamic Bayesian Network to infer the probabilities of different options that the intruder might pursue next. We constructed our Bayesian Net utilizing the open source JavaBayes package. In addition to any route segment matches for the observed trajectory legs (described above), the Bayesian Network has variables representing many pieces of information that might be available about the intruder and area: 1) VFR or IFR; 2) transponder code; 3) aircraft characteristics (e.g. aircraft size, wing type, tail type, number of engines, engine type, observed gear position); 4) communications on relevant frequencies, 5) ceiling and visibility, and 6) wind direction and intensity. Feeding this information as evidence into the Bayesian Network, we are able to infer the probabilities of possible next legs, or other actions by the intruder.

Threat Evaluation: Our trajectory prediction algorithm returns multiple possible medium-term future trajectories with associated probabilities for each aircraft. Threat Evaluation consists of taking these trajectory predictions and detecting possible loss-of-separation violations during the time horizon for the SAA algorithm (30-120secs in our case). We carry this out by performing a fine-grained (1sec) discretization of all of the probabilistic trajectories, mapping them to a 2D data structure (ignoring altitude) and reporting any possible overlaps. Possible overlaps are then filtered to rule out aircraft that will remain separated in altitude.

Threat Resolution: The threat evaluation component returns a set of probabilistic trajectories that potentially violate loss-of-separation thresholds within the time horizon for the SAA algorithm. Threat Resolution consists of computing aircraft maneuvers which will steer the ownship clear of the trajectories that pose threats. This is typically done by choosing from a set of maneuvers that eliminate conflicts and optimize some risk/benefit function. We propose a probabilistic threat resolution approach explained in detail in a subsequent section. Currently, we reuse the approach implemented by the GenericSAA framework, which has the following characteristics: 1) Only the most immediate threat is addressed. 2) Resolutions are computed by examining a small set of maneuvers that modify one of {Horizontal Position, Altitude, Speed}, in that order. 3) The first maneuver that yields a conflict-free trajectory projection is chosen as the solution; no optimization is performed at this point.

We have implemented most of the elements in our Trajectory Prediction architecture: Leg Extraction, Segment Matchings, and Route Prediction using a Dynamic Bayesian Network. For the time being we have manually created Route Database entries for a small test area near Sacramento, CA. We have integrated our probabilistic trajectory prediction with ACES' GenericSAA framework. GenericSAA provides Threat Evaluation and Threat Resolution functionality and therefore allows us to evaluate our approach in the context of a complete SAA solution. For verification purposes we have also built a lightweight simulation and visualization mechanism (Figure 9) that allows us to 1) Set up intruder and ownship flights and 2) Step through time to incrementally visualize the aircraft states, projected trajectories, computed threats and computed threat resolution maneuvers.

5 Improving Probabilistic Path Prediction

As described in the previous section, we developed a prototype dynamic Bayesian Network model that takes advantage of knowledge about aircraft, routes, traffic patterns, topography, airspace, weather information, and observed communications in order to better predict the path of an intruder aircraft. While this additional information can be very powerful, the states in the Bayesian network must be tailored to the specific area of the intruder. For example, if the intruder

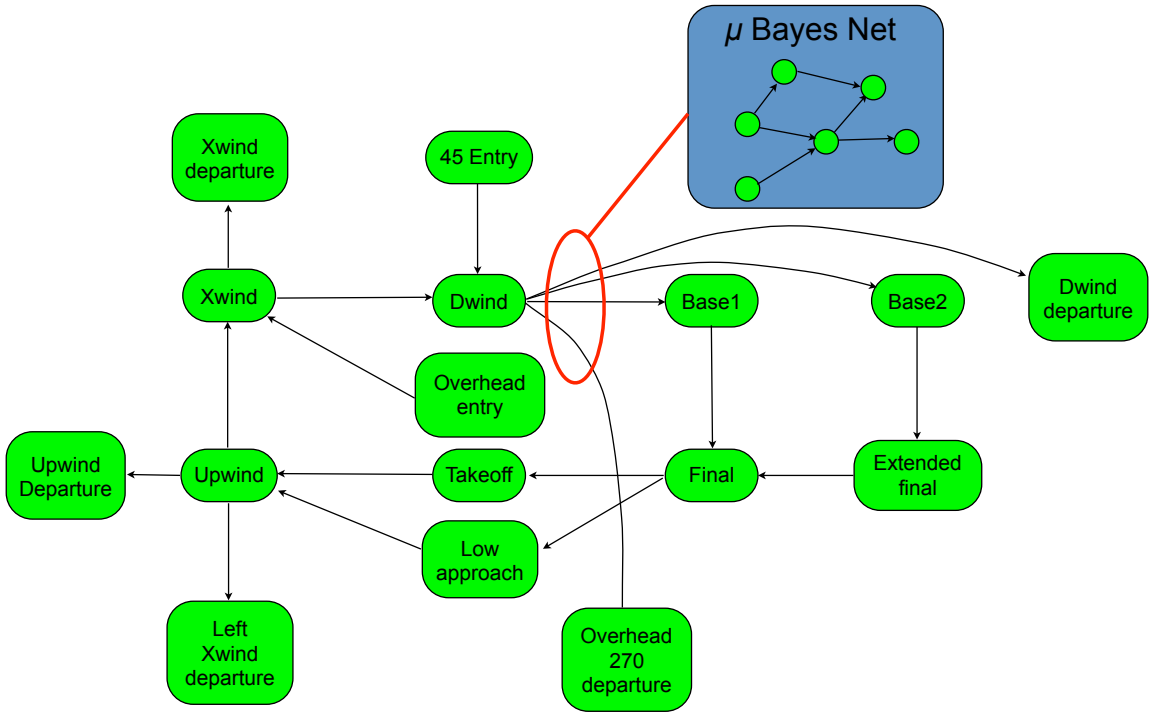


Figure 10: Bayesian Network capturing probabilistically the relation between different characteristics of the intruder aircraft and how they can be used to infer the probability of what the intruder may do next.

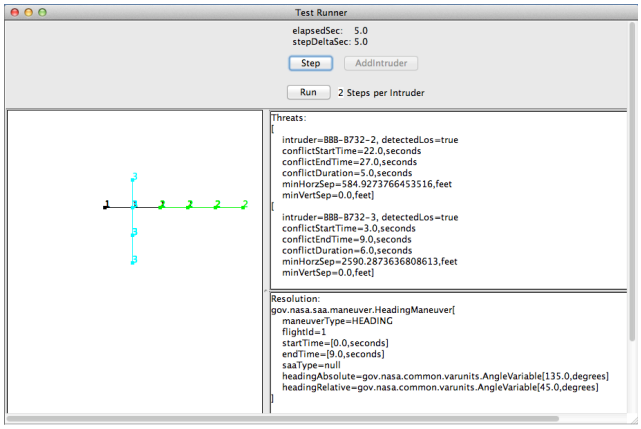


Figure 9: SAA Simulator. Diagram on the left shows relative track of intruder threats (horizontal line) compared with track of the UAS. Each point is a time step. Text on the right describes details of the threats (conflict time, duration, separation), and recommended resolution maneuver.

is near a particular airport, then possible next states for the aircraft may include traffic pattern legs or approach legs for that airport. However, if the aircraft is at 4000 ft AGL, then those legs are not relevant as possible next states for the aircraft. In our current prototype, we dodged this issue by including all possible states for the local area whether or not they were reachable by the intruder within the time window of interest. This resulted in the current and next state nodes in the Bayesian network having more than 30 possible values. As a result, the conditional probability tables for some of these nodes required more than 1000 entries. Although most of the entries in this table are zero, it is impractical to generate these tables by hand for anything other than a limited geographical area. In addition, the conditional probability tables are not very compact or easy to understand. Based on our experience, we believe that it is possible to encode this information much more succinctly, and automatically generate the Bayesian network for a particular location. The key to this is to recognize that traffic patterns and procedures at an airport can be encoded as probabilistic automata. For each leg in the procedure there are a set of possible successor legs with different probabilities. Those probabilities are influenced by factors such as aircraft type, airspeed, vertical speed, transponder code, etc. As a result, each set of transitions for a leg form a small dynamic Bayesian network. Figure 10 provides an illustration of what this automata looks like for a typical airport traffic pattern. An aircraft remaining in the pattern (practicing landings) would transition from Takeoff to Upwind to Crosswind (Xwind) to Downwind

Lateral resolution actions	Vertical resolution actions
Standard rate left turn	Climb 1000 ft/min
Half standard rate left turn	Climb 500 ft/min
Straight	Level
Half standard rate right turn	Descend 500 ft/min
Standard rate right turn	Descend 1000 ft/min

Table 1: Resolution actions.

(Dwind) to Base to Final. There are multiple possible transitions at many of the nodes. For example, from downwind, the aircraft could transition to a close-in base (Base1) a more distant base (Base2), could depart downwind, or do an overhead 270 departure. The probabilities of these different options are influenced by other factors, such as aircraft type, speed, climb/descent, altitude, transponder code, communications, etc. We believe that most of this information can be encoded generically for traffic patterns, approaches, departures, and airways, and that we can automatically generate the appropriate dynamic Bayesian Network for any specific location.

We recognize that our approach to "plan recognition" is rather specialized for aircraft trajectories. We have considered recent approaches such as that of Ramirez and Geffner (2010). The trouble is that for this problem there is little ability to predict the long term goal of the aircraft. Instead, we are interested in predicting the next plan or trajectory steps. In addition, the computational overhead of the Ramirez/Geffner approach is substantial and would likely be impractical for this domain. One can, however, think of the dynamic Bayesian Network as encoding the information that would be obtained from an approach like that of Ramirez/Geffner – the probability of each possible next plan step is influenced by the previous actions of the aircraft, as well as the probability of the possible goals.

6 Medium-term Probabilistic Threat Resolution

Systems such as TCAS and ACAS X are designed to detect and avoid near term conflicts between aircraft conflicts that would occur in a time frame between 15 and 48 seconds. If a Near Mid-Air Collision (NMAC) is imminent (within 25 seconds) they issue a resolution advisory that involves climbing, descending, or constraining vertical speed (e.g. maintain climb, or do not descend). Because these systems deal with near term critical situations, the resolution advisories can be dramatic in nature, and can be disruptive and inefficient for the intended flight path of one or both aircraft.

It is clearly desirable to have medium or longer term conflict detection and avoidance, so that near-term conflict avoidance can be minimized. In particular, we would like to recognize potential conflicts up to two minutes in advance, resolve them using less abrupt maneuvers, and keep the aircraft well clear of each other. Furthermore, for smaller aircraft and UASs operating at lower altitudes, climbing and descending is often not the best approach for resolving conflicts – descent may be limited by terrain or obstacles, and climb may be limited by performance, clouds, or airspace. As a result, most conflicts between VFR aircraft are resolved by heading changes to go around or pass behind the other aircraft.

Our objective is to develop and test an algorithm for medium-term conflict resolution for conflicts that are predicted in the time frame between 30 seconds and 2 minutes. We assume a probabilistic model of the possible pose (location, airspeed, vertical speed, heading, and turn rate) for the intruder aircraft as a function of time. In particular, the model developed and described earlier may predict specific future actions with high probability for the intruder based on inference that the intruder is on a particular route, on an approach, in the traffic pattern, or will need to deviate based on weather, terrain, or airspace constraints. We intend to consider both lateral resolutions (e.g. standard rate left turn), and altitude resolutions (e.g. descend 500 ft/min) as shown in Table 1.

The vertical resolutions in Table 1 are much gentler than those considered by TCAS and ACAS X for two reasons: 1) most small aircraft are limited to sustained climb rates between 1000 ft/min and 1500 ft/min, and 2) because of the longer time horizon, climbs and descents do not need to be aggressive. From the initial state of our aircraft, each vertical and lateral possibility will be considered at each time step, which we will take to be every 15 seconds. Given that there are 6 time steps from 2 minutes until 30 seconds, and 25 action combinations at each time step, this results in a state space of 25^6 or approximately 250 million states. However, the vertical and lateral spaces can be considered independently, resulting in two spaces of approximately 16,000 states each, a much more manageable number.[1] An illustration of the lateral and vertical state spaces are shown in the Figure 11.

There are costs associated with the different actions that the aircraft can take. In particular, there is a cost associated with initiating each new action (other than none), and more aggressive actions cost more than less aggressive actions. Given the distribution of possible projected locations for the intruder, each of the leaf states in this state tree can be given a value based on the clearance from the intruder, and the cost of returning to the original route. Using dynamic programming, this information can be backed up through the state space to find the best action at each time step. This approach, is similar to the approach taken in ACAS X, but there are some differences:

1. The action space is quite different we are considering both lateral and vertical actions, but the actions are less aggressive.
2. The time horizon is much longer, but we use much coarser time steps because the conflict is farther off.

[1] Although we are dealing with a longer time horizon than ACAS X, and lateral as well as vertical resolutions, our discretization of time is much coarser. As a result the search space is actually smaller. We can get away with this because with a longer time horizon it is not necessary to consider action choices at such fine time intervals.

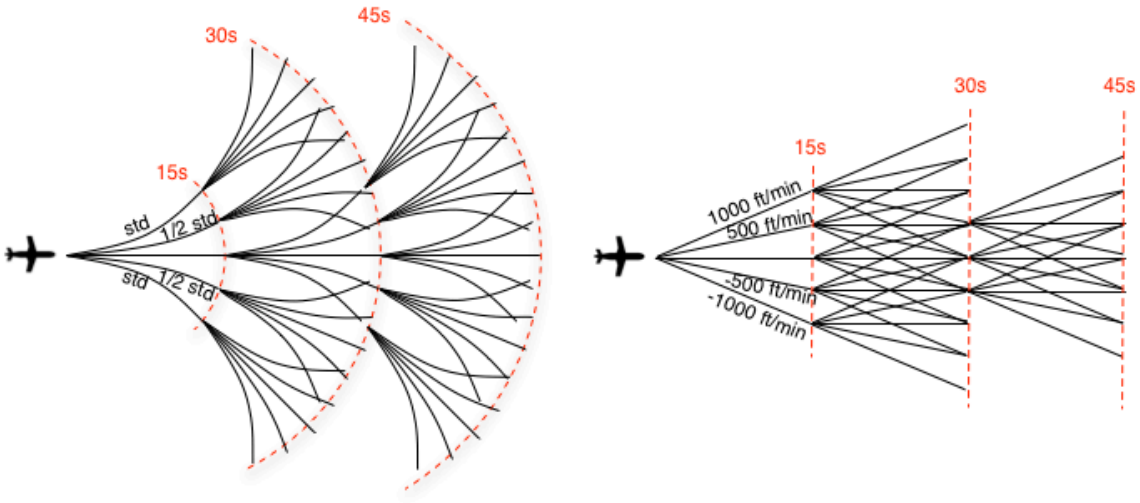


Figure 11: The lateral and vertical action spaces for threat resolution. For lateral maneuvers, standard and half-standard rate turns are considered every 15 seconds resulting in heading changes of 45 and 22.5 degrees respectively. For vertical maneuvers, 500 and 1000 ft/min climb and descent are considered every 15 seconds.

However, the most critical difference is due to our knowledge intensive approach to prediction of the course of the intruder. Because the probability distribution for the intruder is influenced by location (routes, weather, terrain), the values on the leaf nodes of the MDP are dependent on the particular situation. As a result, the MDP cannot be solved in advance and compiled into a decision table as in ACAS X instead it must be solved in real time. Given the size of the MDP, we believe that this can be done in a few seconds. If this turns out not to be the case, we can resort to approximate policy generation methods for solving the MDP, which would reduce solution time. Because of the medium-term time frame, optimal solution of the MDP is not as important as for ACAS X.

One final difference between this approach and that of ACAS X is that the value of each leaf node is also influenced by the cost of returning the UAS to its intended course. Deviating more from the intended course increases the penalty, and this must be balanced with maintaining clearance from the intruder. There is no consideration of this issue in TCAS or ACAS X.

7 Conclusion

There is growing demand to operate UASs in civilian airspace. Enabling this will require much better sense-and-avoid technology, so that UASs of vastly different sizes and capabilities can maintain separation from other aircraft. While most current research has concentrated on near-term collision detection and avoidance utilizing only the observed trajectory history of an aircraft, we target mid-term detection and avoidance utilizing a variety of information beyond just aircraft trajectory.

We concentrated on two tasks: (1) conflict detection: utilizing probabilistic path recognition; and (2) conflict resolution: utilizing fast online probabilistic path planning. Our work is preliminary. We have implemented the components necessary for the conflict detection portion and integrated it with the ACES traffic simulation system. We are refining and improving the dynamic Bayesian Network prediction and plan to implement our probabilistic conflict resolution approach. We hope to evaluate the techniques using a large database of actual air traffic trajectories that have been collected and made available through related efforts at NASA.

8 Acknowledgements

We would like to thank Eric Mueller for helping us hone our ideas on this project. We would like to thank Confesor Santiago for help with the ACES simulation environment, and the GenericSAA module developed for ACES. This work was supported by the NASA Aeronautics Research Institute (NARI).

References

Airspace Systems Division, NASA Ames Research Center. Airspace Concept Evaluation System (ACES). http://www.aviationsystemsdivision.arc.nasa.gov/research/modeling/aces.shtml.

Barnhart, R.; Hottman, S.; Marshall, D.; and Shappee, E., eds. 2012. *Introduction to Unmanned Aircraft Systems*. CRC Press.

Chen, W.-Z.; Wong, L.; Kay, J.; and Raska, V. 2009. Autonomous sense and avoid (SAA) for unmanned air systems (UAS). In *Systems Concepts and Integration Panel (SCI) Symposium*, volume RTO-MP-SCI-202. NATO Science and Technology Organization. ftp://ftp.rta.nato.int/PubFullText/RTO/MP/RTO-MP-SCI-202/MP-SCI-202-28.doc.

Federal Aviation Administration. 2009. Literature review on detect, sense, and avoid technology for unmanned aircraft systems. Technical Report DOT/FAA/AR-08/41, U.S. Department of Transportation. http://www.tc.faa.gov/its/worldpac/techrpt/ar0841.pdf.

Federal Aviation Administration. 2011. TCAS II, Version 7.1. Technical Report HQ 111358, U.S. Department of Transportation. http://www.faa.gov/documentLibrary/media/Advisory_Circular/TCAS II V7.1 Intro booklet.pdf.

Francis DiLego, J.; Hitchings, J.; Salisbury, C.; Simmons, H.; Sterling, J.; and Cai, J. 2009. Joint Airspace Management and Deconfliction (JASMAD). Technical Report AFRL-RI-RS-TR-2009-13, Air Force Research Laboratory.

Kochenderfer, M.; Holland, J.; and Chryssanthacopoulos, J. 2012. Next-generation airborne collision avoidance system. *Lincoln Laboratory Journal* 19(1):17–33.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 1121–1126. AAAI Press.

AI-MIX: Using Automated Planning to Steer Human Workers Towards Better Crowdsourced Plans

Lydia Manikonda Tathagata Chakraborti Sushovan De
Kartik Talamadupula Subbarao Kambhampati
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287 USA
{lmanikon, tchakra2, sushovan, krt, rao} @ asu.edu

Abstract

One subclass of human computation applications are those directed at tasks that involve planning (e.g. tour planning) and scheduling (e.g. conference scheduling). Interestingly, work on these systems shows that even primitive forms of automated oversight on the human contributors helps in significantly improving the effectiveness of the humans/crowd. In this paper, we argue that the automated oversight used in these systems can be viewed as a primitive automated planner, and that there are several opportunities for more sophisticated automated planning in effectively steering the crowd. Straightforward adaptation of current planning technology is however hampered by the mismatch between the capabilities of human workers and automated planners. We identify and partially address two important challenges that need to be overcome before such adaptation of planning technology can occur: (i) *interpreting* inputs of the human workers (and the requester) and (ii) *steering* or critiquing plans produced by the human workers, armed only with *incomplete* domain and preference models. To these ends, we describe the implementation of **AI-MIX**, a tour plan generation system that uses automated checks and alerts to improve the quality of plans created by human workers; and present a preliminary evaluation of the effectiveness of steering provided by automated planning.

1 Introduction

In solving computationally hard problems – especially those that require input from humans, or for which the complete model is not known – human computation has emerged as a powerful and inexpensive approach. One such core class of problems is *planning*. Several recent efforts have started looking at crowd-sourced planning tasks (Law and Zhang 2011; Zhang et al. 2012; 2013; Lasecki et al. 2012; Lotosh, Milo, and Novgorodov 2013). Just like in a formal organization, the quality of the resulting plan depends on effective leadership. We observe that in most of these existing systems, the workers are steered by primitive automated components that merely enforce checks and ensure satisfaction of simple constraints. Encouragingly, experiments show that even such primitive automation improves plan quality, for little to no investment in terms of cost and time.

This begs the obvious question: *is it possible to improve the effectiveness of crowdsourced planning even further by using more sophisticated automated planning technologies?* It is reasonable to expect that a more sophisticated automated planner can do a much better job of steering the crowd (much as human managers "steer" their employees). Indeed, work such as (Law and Zhang 2011) and (Zhang et al. 2012) is replete with hopeful references to the automated planning literature. There exists a vibrant body of literature on automated plan generation, and automated planners have long tolerated humans in their decision cycle – be it mixed initiative planning (Ferguson, Allen, and Miller 1996) or planning for teaming (Talamadupula et al. 2010). The context of crowdsourced planning scenarios, however, introduces a *reversed mixed initiative planning* problem – the planner must act as a guide to the humans, who are doing the actual planning. The humans in question can be either experts who have a stake in the plan that is eventually created, or crowd workers demonstrating collective intelligence.

In this paper, we present **AI-MIX** (Automated Improvement of Mixed Initiative eXperiences), a new system (Manikonda et al. 2014) that implements a general architecture for human computation systems aimed at planning and scheduling tasks. **AI-MIX** foregrounds the types of roles an automated planner can play in such systems, and the challenges involved in facilitating those roles. The most critical challenges include:

Interpretation: Understanding the requester's goals as well as the crowd's plans from semi-structured or unstructured natural language input.

Steering with Incompleteness: Guiding the collaborative plan generation process with the use of incomplete models of the scenario dynamics and preferences.

The *interpretation* challenge arises because human workers find it most convenient to exchange / refine plans expressed in a representation as close to natural language as possible, while automated planners typically operate on more structured plans and actions. The challenges in *steering* are motivated by the fact that an automated planner operating in a crowdsourced planning scenario cannot be expected to have a complete model of the domain and the preferences; if it does, then there is little need or justification for using human workers! Both these challenges are further complicated by the fact that the (implicit) models used by the human workers and the automated planner are very likely to differ in many ways, making it challenging for the planner to critique the plans being developed by the human workers.

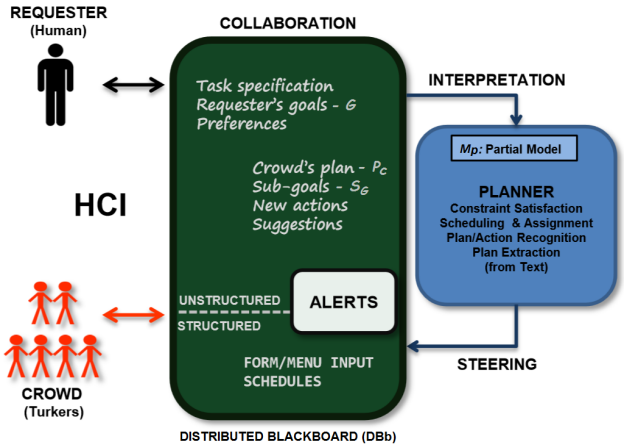


Figure 1: A generalized architecture for crowdsourced planning systems.

In the rest of the paper we describe how these challenges are handled in AI-MIX, and present an evaluation of their effectiveness in steering the crowd. The paper is organized as follows. We first look at the problem of planning *for* crowdsourced planning in more detail, and present a generalized architecture for this task. Next, we consider the roles that an automated planner can play within such an architecture, and discuss the challenges that need to be tackled in order to facilitate those roles. We then describe **`AI-MIX`** and present a preliminary evaluation of the effectiveness of the steering provided by automated planning on Amazon's MTurk platform[1]. We hope that this work will spur more directed research on the challenges that we have identified.

2 Planning for Crowdsourced Planning

The crowdsourced planning problem involves constructing a plan from a set of activities suggested by the crowd (henceforth referred to as the turkers) as a solution to a task, usually specified by a user called the *requester*. The requester provides a high-level description of the task – most often in natural language – which is then forwarded to the *turkers*. The turkers can perform various roles, including breaking down the high-level task description into more formal and achievable sub-goals (Law and Zhang 2011), or adding actions into the plan that support those sub-goals (Zhang et al. 2012). The term *planner* is used to refer to the automated component of the system, and it performs various tasks ranging from constraint checking, to optimization and scheduling, and plan recognition. The entire planning process must itself be iterative, proceeding in several rounds which serve to refine the goals, preferences and constraints further until a satisfactory plan is found. A general architecture for solving this crowdsourced planning problem is depicted in Figure 1.

2.1 Roles of the planner

The planning module, or the automated component of the system, can provide varying levels of support. It accepts both the sub-goals S_G, and crowd's plan P_C, as input from the turkers. This module analyzes the current plan generated by the crowd, as well as the sub-goals, and determines constraint and precondition violations according to the model M_P of the task that it has. The planner's job is to steer the crowd towards more effective plan generation.

However, the three main actors – turkers, requester, and planner – need a common space in which to interact and exchange information. This is achieved through a common interactive space – the *Distributed Blackboard* (DBb) – as shown in Figure 1. The DBb acts as a collaborative space where information related to the task as well as the plan that is currently being generated is stored, and exchanged between the various system components.

In contrast to the turkers, the planner cannot hope for very complex, task-specific models, mostly due to the difficulty of creating such models. Instead, a planner's strong-suit is to automate and speed-up the checking of plans against whatever knowledge it *does* have. With regard to this, the planner's model M_P can be considered shallow with respect to preferences, but may range the spectrum from shallow to deep where domain physics and constraints are concerned (Zhuo, Kambhampati, and Nguyen 2012). The planning process itself continues until one of the following conditions (or a combination thereof) is satisfied:

- The crowd plan P_C reaches some satisfactory threshold and the requester's original goal G is fulfilled by it; this is a subjective measure and is usually determined with the intervention of the requester.
- There are no more outstanding alerts, and all the sub-goals in S_G are supported by one (or more) actions in P_C.

3 Planning Challenges

From the architecture described in Figure 1, it is fairly obvious that a planner (automated system) would interact with the rest of the system to perform one of two tasks: (1) **interpretation** and (2) **steering**.

Interpretation is required for the planner to inform itself about what the crowd *is* doing; steering is required for the planner to tell the crowd what they *should* be doing.

3.1 Interpretation of the Crowd's Evolving Plan

The planner must interpret the information that comes from the requester, and from the crowd, in order to act on that information. There are two ways in which the planner can ensure that it is able to understand that information:

Force Structure The system can enforce a pre-determined structure on the input from both the requester, and the crowd. This can by itself be seen as part of the model M_P, since the planner has a clear idea about what kind of information can be expected through what channels. The obvious disadvantage is that this reduces flexibility for the turkers. In the tour planning scenario (our main application domain that we explain in Section. 4), for example, we might force the requester to number his/her goals, and force the turkers to explicitly state which goals their proposed plan aims to handle (c.f. (Zhang et al. 2012)). The turkers could also be required to add other structured attributes to their plans such as the duration and cost of various activities (actions).

[1] Amazon Mechanical Turk, http://www.mturk.com

Extract Structure The planner can also *extract* structure from the turker inputs to look for specific action descriptions that are part of the planner's model M_P, in order to understand what aims a specific plan is looking to achieve. Although this problem has connections to plan recognition (Ramírez and Geffner 2010), it is significantly harder as it needs to recognize plans not from actions, but rather textual descriptions. Thus it can involve first recognizing actions and their ordering from text, and then recognizing plans in terms of those actions. Unlike traditional plan recognition that starts from observed plan traces in terms of actions or actions and states, the interpretation involves first extracting the plan traces. Such recognition is further complicated by the impedance mismatch between the (implicit) planning models used by the human workers, and the model available to the planner.

Our system uses both the techniques described above to gather relevant information from the requester and the turkers. The requester provides structured input that lists their constraints as well as goals (and optionally cost and duration constraints), and can also provide a free unstructured text description for the task. The turkers in turn also provide semi-structured data - they are given fields for activity title, description, cost and duration. The turkers can also enter free text descriptions of their suggestions; the system can then automatically extract relevant actions by using Natural Language Processing (NLP) methods to match the input against the planner's model M_P.

3.2 Steering the Crowd's Plan

There are two main kinds of feedback an automated planner can provide to the human workers:

Constraint Checking One of the simplest ways of generating helpful suggestions for the crowd is to check for quantitative constraints imposed by the requester that are violated in the suggested activities. In terms of the tour planning scenario, this includes: (i) cost of a particular activity; and (ii) the approximate duration of an activity. If the requester provides any such preferences, our system is able to check if they are satisfied by the crowd's inputs.

Constructive Critiques Once the planner has some knowledge about the plan that the turkers are trying to propose (using the extraction and recognition methods described above), it can also try to actively help the creation and refinement of that plan by offering suggestions as part of the alerts. These suggestions can vary depending on the depth of the planner's model. Some examples include: (i) simple notifications of constraint violations, as outlined previously; (ii) plan critiques (such as suggestions on the order of actions in the plan and even what actions must be present); (iii) new plans or plan fragments because they satisfy the requester's stated preferences or constraints better; (iv) new ways of decomposing the current plan (Nau et al. 2003); and (v) new ways of decomposing the set of goals S_G.

4 System Description

The following section describes in detail the **AI-MIX** system that was deployed on Amazon's MTurk platform to engage the turkers in a tour planning task. The system is similar to Mobi (Zhang et al. 2012) in terms of the types of inputs it can handle and the constraint and quantity checks that it can provide (we discuss this further in Section 5.1). However, instead of using structured input, which severely restricts the turkers and limits the scope of their contributions, our system is able to parse natural language from user inputs and reference it against relevant actions in a domain model. This enables more meaningful feedback and helps provide a more comprehensive tour description.

4.1 Requester Input

The task description, as shown in Figure 2, is provided by the requester in the form of a brief description of their preferences, followed by a list of activities they want to execute as part of the tour, each accompanied by a suitable hashtag. For example, the requester might include one dinner activity and associate it with the tag `#dinner`. These tags are used internally by the system to map turker suggestions to specific tasks. The upper half of Figure 2 shows an example of a requester task, which includes a block of text for the turkers to extract context from, and structured task requests associated with hashtags.

4.2 Interface for Turkers

In addition to the task description, the **AI-MIX** interface also contains a section that lists instructions for successfully submitting a Human Intelligence Task (HIT) on Amazon MTurk. HIT is the individual task that the turkers work on, in this context consisting of either adding an action or a critique, as discussed in more detail later. The remaining components, arranged by their labels in the figure, are:

1. **Requester Specification**: This is the list of requests and to-do items that are yet to be satisfied. All the unsatisfied constituents of this box are initially colored red. When a tag receives the required number of supporting activities, it turns from red to green. Tags that originated from the requester are classified as top-level tags, and are always visible. Tags that are added by the automated planner or by turkers are classified as lower priority, and disappear once they are satisfied by a supporting activity.
2. **Turker Inputs**: Turkers can choose to input one of two kinds of suggestions: (i) a new action to satisfy an existing to-do item; or (ii) a critique of an existing plan activity.
3. **Turker Responses**: The "Existing Activities" box displays a full list of the current activities that are part of the plan. New turkers may look at the contents of this box in order to establish the current state of the plan. This component corresponds to the *Distributed Blackboard* mentioned in Section 2.1.
4. **Planner Critiques**: The to-do items include automated critiques of the current plan that are produced by the planner. In the example shown, "broadwayshow_showing" is a planner generated to-do item that is added in order to improve the quality of the turkers' plan.

Finally, the right hand portion of the interface consists of a map, which can be used by turkers to find nearby points of interest, infer routes of travel or the feasibility of existing

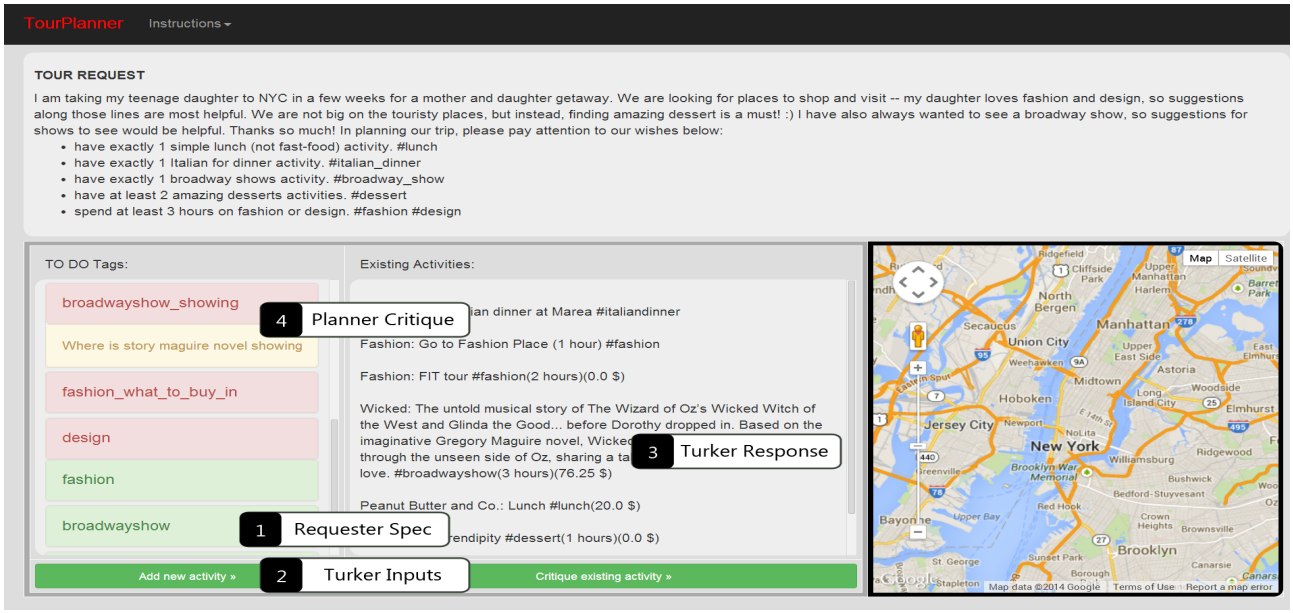


Figure 2: The **AI-MIX** interface showing the distributed blackboard through which the crowd interacts with the system.

suggestions, or even discover new activities that may satisfy some outstanding tags.

Activity Adddition The "Add Activity" form is shown in Figure 3. Turkers may choose to add as many new activities as they like. Each new activity is associated with one of the to-do tags. After each activity is submitted, a quantitative analysis is performed where the activity is (i) checked for possible constraint (duration or cost) violations; or (ii) critiqued by the planner.

Action Extraction In order to extract meaning from the new activities described by the turkers, the system performs parts of speech (PoS) tagging on the input text using the *Stanford Log-Linear Part-of-Speech* tagger (Toutanova et al. 2003). It identifies the name of the suggested activity and places that the turkers are referring to using the verb and noun parts of the tagger's output respectively.

Sub-Goal Generation **AI-MIX** uses the same tags used by turkers while inputting activities in order to determine whether the planner has additional subgoal annotations on that activity. To facilitate this, the planner uses a primitive PDDL (McDermott et al. 1998) domain description of general activities that may be used in a tour-planning applications – this description corresponds to the *planner model* M_P introduced earlier. Examples of actions in M_P include high level activities such as `visit`, `lunch`, `shop` etc. Each activity is associated with a list of synonyms, which helps the planner in identifying similar activities. Currently, we generate these synonyms manually, but it is possible to automate this via the use of resources such as WordNet. Each action also comes with some generic preconditions. When the planner determines that a turker generated activity matches one of the actions from its model, it generates subgoals to be added as to-do items back in the interface based on the preconditions of that action. An example of an action description (for the "visit" action) is given below:

```
(:action visit ;; synonyms: goto, explore
 :parameters (?p - place)
 :precondition (at ?p) ;; Getting to ?p,
 ;; Entrance fee ?p, ;; Visiting hours ?p
 :effect (visited ?p))
```

In the example given above, the planner would pop up the three preconditions – `Getting to`, `Entrance fee`, and `Visiting hours` – as to-do sub-goals for any `visit` actions suggested by turkers. The system also provides some helpful text on what is expected as a resolution to that to-do item – this is indicated by the yellow "planner critique" box in Figure 2.

Constraint Checking In addition to generating sub-goals for existing activities, our system also automatically checks if constraints on duration and cost that are given by the requester are being met by the crowd's plan. If these constraints are violated, then the violation is automatically added to the to-do stream of the interface, along with a description of the constraint that was violated. Turkers can then choose to add an action that resolves this to-do item using the normal procedure.

Adding Turker Critiques The turkers can also add *critiques* of the actions in the existing plan. To do this, they use the form shown in the lower half of Figure 3. The turkers click on an existing activity, and enter the note or critique in a text box provided. Additionally, they are also asked to enter a child tag, which will be used to keep track of whether an action has been added to the plan that resolves this issue. Turkers can add as many critiques as they want.

Though the current system uses only a preliminary form of automated reasoning, this effort can be seen as the first

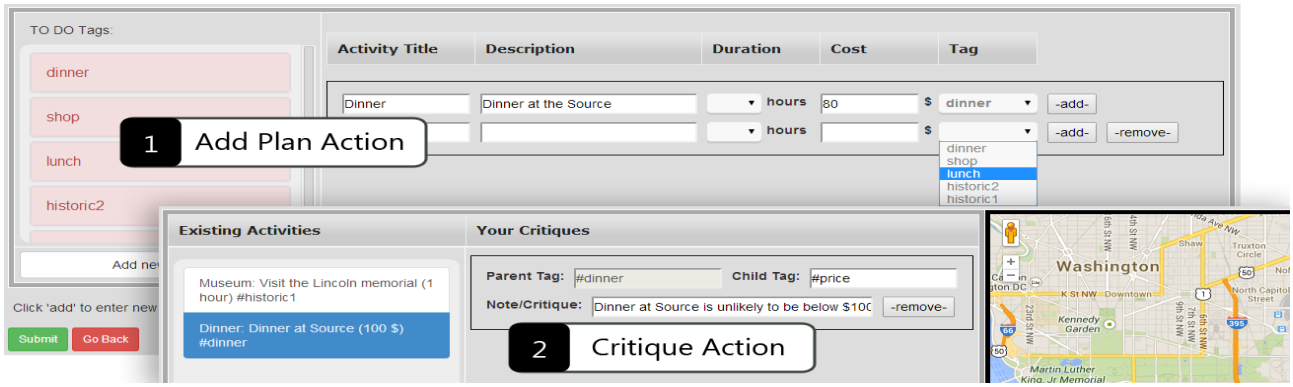


Figure 3: Adding and critiquing activities (plan actions) in the **AI-MIX** system.

step towards incorporating more sophisticated methods for plan recognition and generation (Talamadupula and Kambhampati 2013). A video run-through of our system can be found at the following URL: http://youtu.be/73g3yHClx90.

5 Experiments

5.1 Experimental Setup

For our study, HITs were made available only to the turkers within US (since the requests involved locations inside the US) with a HIT approval rate greater than 50%. Turkers were paid 20 cents for each HIT, and each turker could submit 10 HITs per task. We used tour planning scenarios for six major US cities, reused from the Mobi system's evaluation (Zhang et al. 2012). To measure the impact of automated critiquing on the generated plans, we compared the results from three experimental conditions:

C1: Turkers could give suggestions in free text after reading the task description - there were no automated critiques.

C2: Turkers quantified their suggestions in terms of cost and duration, and the system checked these constraints for violations with respect to the requester demands.

C3: In addition to C2, the system processed free-form text from turker input, and extracted actions to match with our planning model in order to generate alerts for sub-goals and missing preconditions.

C1 and C2 were compared to the proposed approach, C3, separately. Each set was uploaded at the same time, with the same task description and HIT parameters. In the first run, C3 and C2 were compared on 6 scenarios (New York, Chicago, San Francisco, Las Vegas, Washington and Los Angeles) and were given 2 days before the HITs were expired. The interfaces for both C3 and C2 were made identical to eliminate any bias. In the second run, the conditions C1 and C3 were run over a period of one day, for the two scenarios which were most popular in the first run (New York and Chicago). For each of these tasks, the requester prepopulated the existing activities with one or two dummy inputs that reflect the kinds of suggestions she was looking for. In sum, we had more than 150 turkers who responded to our HITs. The analysis that follows is from the 35 turkers who contributed to the final comparisons among C1, C2, and C3.

5.2 Task Completion Latency

When C3 was compared to C1 over a period of one day, we found that C3 received four responses from 3 distinct turkers, whereas C1 failed to attract any responses. This might indicate that the presence of the "TO DO" tags generated by the automated critiquing component was helpful in engaging the turkers and guiding them towards achieving specific goals. However, there may also be alternate explanations for the fact that C1 did not receive any inputs, such as turker fatigue, or familiarity with the C3 interface from previous runs. There is need for further experimentation before these results can be conclusively proved.

We also looked at the number of HITs taken to complete the tasks for each of the scenarios. After the HITs were expired, none of the tasks were entirely complete (a task is "completed" if there are no more outstanding to-do items), but C2 had 3.83 unfulfilled tags per HIT as compared to 10.5 for C3. As expected, the task completion latency seems to have increased for C3, since alerts from the system drive up the number of responses required before all the constraints are satisfied. However, as shown next, the increased quality of generated plans may justify this additional latency.

5.3 Generated Tour Plan Quality

We see that the quality of the plans, in terms of detail and description, seems to increase in C3, since we now have users responding to planner critiques to further qualify suggested activities. For example, a turker suggested "not really fun, long lines and can not even go in and browse around" in response to a planner generated tag (related to a "fun club" activity suggested previously), while another suggested a "steamer" in response to a planner alert about "what to eat for lunch". A comparison between the plans generated for C2 and C3 (for New York City) is given in Table 1. This seems to indicate that including a domain description in addition to the simplistic quantity and constraint checks increases the plan quality.

5.4 Role Played by the Planner Module

We now look at some statistics that indicate the role played by the automated module in the tasks. We received a total of 31 new activity suggestions from turkers, of which 5

Show: Go to TKTS half ticket discount booth. You have to stand in line early but it's an authentic nyc experience #show(3 hours)(200.0 $)
Show: Go to show #show(3 hours)(200.0 $)
Show: ABSOLUTELY CANNOT go wrong with Phantom of the Opera #show(3 hours)(200.0 $)
Lunch: Alice's Tea Cup #lunch(20.0 $)
Design: Walk around the Garment District (go into shops) just south of Times Square. They often print their own fabrics. #design(2 hours)(0.0 $)
Dessert: Serendipity #dessert(1 hours)(10.0 $)
piccolo angolo: Italian in the Village - real deal #italiandinner(2 hours)(60.0 $)
Lombardi's Pizza: #italian_dinner #italiandinner_todo1
Ice Cream: http://www.chinatownicecreamfactory.com/ #italiandinner_todo0
#lunch: Mangia Organics #lunch_todo0
watch Wicked (musical): Do watch Wicked the musical. It's a fantastic show and one of the most popular on Broadway right now! #broadwayshow(3 hours)(150.0 $)
watch How to Succeed in Business: Also a great show, a little less grand than Wicked. #broadwayshow(3 hours)(150.0 $)
Activity Steamer: #lunch #lunch_todo1
Paradis To-Go: Turkey & Gruyere is pretty delicious. The menu is simple, affordable, but certainly worth the time #lunch(1 hours)(10.0 $)
cupcakes!: Magnolia Bakery on Bleecker in the Village #dessert(1 hours)(10.0 $)

Table 1: Sample activity suggestions from turkers for the two conditions: C2 (top) and C3 (bottom).

violated quantity constraints. The C3 interface attracted 39 responses, compared to 28 for C2, which may indicate that the planner tags encouraged turker participation.

Note that in the **AI-MIX** interface, there is no perceptual difference between the critiques generated by the planner and the critiques suggested by humans. With this in mind, there were 8 flaws pointed out by humans, but none were acted upon by other turkers; the planner on the other hand generated 45 critiques, and 7 were acted upon and fixed by turkers. This seems to indicate that turkers consider the planner's critiques more instrumental to the generation of a high quality plan than those suggested by other turkers. Though these results are not entirely conclusive, and might also be attributed to possibilities like the critiques of the planner being more popular because they might have been easier to solve; there is enough evidence to suggest that the presence of an automated system does help to engage and guide the focus of the crowd.

6 Conclusion

In this paper, we presented a system, **AI-MIX**, that is a first step towards using an automated planner in a crowdsourced planning application. We identified two major challenges in achieving this goal: *interpretation* and *steering*. We then described the framework of **AI-MIX**, and showed how these challenges were handled by our system – using forced structure and structure extraction for interpreting actions; and using constraint checking and automated planner critiques for steering. We also presented preliminary empirical results over the tour planning domain, and showed that using an automated planner results in the generation of better quality plans. Interestingly, it is possible to improve the completeness of the domain model of a planner over time (Yang, Wu, and Jiang 2007). We are continuing to run experiments using more scenarios and larger time scales to provide further validation for our hypotheses. We are also looking at the problem of eliciting information (Kaplan et al. 2013) from the crowd in order to go from the current list of activities suggested by the crowd, to a more structured *plan* in the traditional sense of the word.

Acknowledgments. This research is supported in part by the ARO grant W911NF-13-1-0023, the ONR grants N00014-13-1-0176 and N0014-13-1-0519, and a Google Research Grant.

References

Ferguson, G.; Allen, J.; and Miller, B. 1996. Trains-95: Towards a mixed-initiative planning assistant. In *Proc. of AIPS-96*, 70–77.

Kaplan, H.; Lotosh, I.; Milo, T.; and Novgorodov, S. 2013. Answering planning queries with the crowd. *In Proc. of VLDB Endowment* 6(9):697–708.

Lasecki, W. S.; Bigham, J. P.; Allen, J. F.; and Ferguson, G. 2012. Real-time collaborative planning with the crowd. In *Proc. of AAAI*.

Law, E., and Zhang, H. 2011. Towards large-scale collaborative planning: Answering high-level search queries using human computation. *In Proc. of AAAI*.

Lotosh, I.; Milo, T.; and Novgorodov, S. 2013. CrowdPlanr: Planning Made Easy with Crowd. In *Proc. of ICDE*. IEEE.

McDermott, D.; Knoblock, C.; Veloso, M.; Weld, S.; and Wilkins, D. 1998. PDDL–the Planning Domain Definition Language: Version 1.2. *Yale Center for Computational Vision and Control, Tech. Rep. CVC TR-98-003/DCS TR-1165*.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An HTN planning system. *JAIR* 20:379–404.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proc. of AAAI*.

Talamadupula, K., and Kambhampati, S. 2013. Herding the crowd: Automated planning for crowdsourced planning. *CoRR* abs/1307.7720.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for human-robot teaming in open worlds. *TIST* 1(2):14.

Toutanova, K.; Klein, D.; Manning, C. D.; and Singer, Y. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic dependency network. In *Proc. of HLT-NAACL*.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence Journal*.

Zhang, H.; Law, E.; Miller, R.; Gajos, K.; Parkes, D.; and Horvitz, E. 2012. Human Computation Tasks with Global Constraints. In *Proc. of CHI*, 217–226.

Zhang, H.; Andre, P.; Chilton, L.; Kim, J.; Dow, S. P.; Miller, R. C.; MacKay, W.; and Beaudouin-Lafon, M. 2013. Cobi: Communitysourcing Large-Scale Conference Scheduling. In *CHI Interactivity 2013*.

Zhuo, H. H.; Kambhampati, S.; and Nguyen, T. A. 2012. Model-lite case-based planning. *CoRR* abs/1207.6713.

Manikonda, L.; Chakraborti, T.; De, S.; Talamadupula, K.; Kambhampati, S. 2014. AI-MIX: Using Automated Planning to Steer Human Workers Towards Better Crowd Sourced Plans. To appear in *Proc. of IAAI*.

A machine learning surrogate for rotorcraft noise optimization

K. Brent Venable
Tulane University and IHMC, USA
kvenabl@tulane.edu

Robert A Morris
NASA Ames, USA
robert.a.morris@nasa.gov

Matthew Johnson
IHMC, USA
mjohnson@ihmc.us

Aliyeh Mousavi
Stanford University, USA
amousavi018@gmail.com

Nikunj Oza
NASA Ames, USA
nikunj.c.oza@nasa.gov

Abstract

Recent increase in interest in using rotorcraft (helicopters and tilt-rotor craft) for public transportation has spurred research in making rotorcraft less noisy, particularly as they land. The ground noise associated with landing trajectories followed by rotorcraft depends in part on the changes in altitude and velocity of the rotorcraft during flight. Models of ground noise taking these altitude and velocity effects into account can be used in an optimization process to determine a set of pilot operations that will lead to noise-minimal landing paths. The Rotorcraft Noise Model (RNM) is one model currently used to analyze rotorcraft landing paths and determine the area-averaged ground noise they produce. However, this model is very slow and inefficient in the optimization process. This project aims to explore a machine learning method, namely Gaussian Processes, to produce faster, but approximate, noise models that can replace the RNM. The experimental results we provide suggest that machine learning has the potential to offer useful approximations to complex engineering systems for evaluating the cost of design decisions, specifically, the design of paths using planners like A* and probabilistic road maps.

Introduction

Recently, there has been greater interest within the aeronautics community in using rotorcraft for commercial air transportation. A significant obstacle to implementing this type of transportation that in large part has deterred commercial rotorcraft use is the significant amount of ground noise produced during landing. While noise is very difficult to accurately model, it does have some non-linear dependence on the changes in altitude and velocity of the rotorcraft during flight. On the most basic level, ground noise increases as a result of an decrease in altitude or an increase in velocity. These phenomena are illustrated in the time-lapse heat maps shown in Figures 1 and 2. Nevertheless, the complexity of the relationship between maneuvering and noise leaves ample margins of improvement with respect to pilots' rule of thumbs which have been employed in the past.

The relationships between ground noise, altitude, and velocity can be used to develop a model that predicts the amount of ground noise produced by different rotorcraft. This model can be interfaced with optimization search algorithms from Artificial Intelligence in order to determine a set of pilot operations that will lead to noise-minimal landing paths. The general approach for this optimization procedure is shown in Figure 3. The search procedure begins by iteratively generating feasible landing paths. These paths and the relevant information about them are then input into a noise model, which then produces an estimate of the amount of ground noise produced by the paths. The optimizer uses the cost information to redirect the search in the direction of quieter paths, until a termination condition is reached. Standard optimization algorithms which have been used include A* (Lindsay, Morris, and Venable 2012), Stochastic Local search (SLS) (Morris et al. 2012) and Probabilistic Road

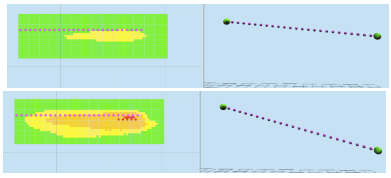


Figure 1: Contour plot and descending pattern representing the effect of change in altitude on ground noise in a landing approach taking place from left to right. In the contour plots, orange and red colors mean more noise.

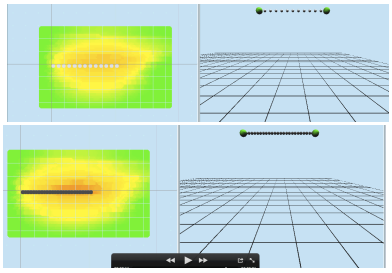


Figure 2: Effect of change in velocity on ground noise on ground noise in a landing approach taking place from left to right. On the left the contour plots are shown, while on the left an increased density of dots represents an increased velocity.

Maps (PRM) (Morris et al. 2013).

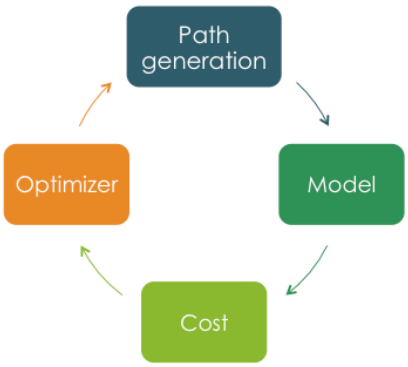


Figure 3: Optimization Approach.

One model currently used to analyze rotorcraft landing paths and determine the area-averaged ground noise they produce is the Rotorcraft Noise Model (RNM) (RNM 2007). This simulation tool developed at NASA Langley Research Center utilizes a vehicle source noise database, experimental measurements, and models in order to simulate how rotorcraft noise propagates through the atmosphere and is ultimately perceived on the ground. While relatively accurate, the RNM is very slow and thus degrades the optimization process due to its inefficiency. The goal of this work is to explore a method from machine learning for replacing the RNM with a faster, but approximate, noise model. As shown in the experimental section, this surrogate model improves the performance of the optimization process by allowing for faster evaluation of candidate paths than is provided by running RNM.

Background

Rotorcraft Noise and Noise Simulation

Helicopter noise sources include the main rotor, the tail rotor, the engine(s), and the drive systems. The most noticeable acoustical property of helicopters is referred to as BVI (Blade Vortex Interaction) noise. This impulsive noise occurs during high-speed forward flight as a result of blade thickness and compressible flow on the advancing blade. A common noise measure is the *Sound Exposure Level (SEL)*. *SEL* provides a comprehensive way to describe noise events for use in modeling and comparing noise environments. The average SEL value over the plane is called the SEL average (SEL_{av}). The equation for SEL_{av} is

$$SEL_{av} = 10log_{10}\Sigma_n(\Sigma_i 10^{SPL_{dB,i,n}/10}\Delta t_{i,n}/T_0)\Delta A_n/A_0 \quad [1]$$

Here, n ranges over the locations on the ground plane and i refers to path elements. $SPL_{db,i,n}$ refers to the *Sound Pressure Level* in decibels for a location and a path element, and the Δs are elemental ground- or trajectory elements. A_0 refers to the area of the ground plane and T_0 is a reference interval of one second.

One challenge in performing a systematic study of approach trajectories for optimization is the cost of verifying results. The most accurate means of verification is through field tests, but these are too costly and time-consuming to perform on a casual basis. Fortunately, there are a number of robust noise models that allow for the evaluation of trajectories through simulation. One such modeling tool is the Rotorcraft Noise Model (RNM) (RNM 2007), a simulation program that predicts how the sound of a rotorcraft propagates through the atmosphere and accumulates on the ground. The core of the RNM method is a database of vehicle source noises defined as *sound spheres*. Spheres are obtained through measured test data or through models. The spheres allow for a representation of the 3D noise directivity patterns associated with the operating rotorcraft. A sphere is associated with one noise source and one flight condition (a value for flight path angle and airspeed). Each sphere represents constant airspeed conditions for a given flight path angle. The sound source properties are extracted from the sphere database using a linear interpolation of both required speed and flight path angle.

The input to RNM consists of a set of computational parameters, including identity of rotorcraft, and the dimensions and resolution of a grid that will display output noise and a specification of the flight trajectory, including position, velocity and orientation. RNM outputs a time history and the effective SEL (or similar metrics) at any location along the path. The result is usually displayed as a contour plot (Figure 4) over a ground plane. Each color corresponds to a dB level (redder and lighter colors noisier). One important parameter that controls performance of RNM is the grid resolution. The grid resolution specifies the distance between grid points, and thus the size of the area of the ground surface (in the SEL computation) being evaluated. Higher resolution means more accurate prediction of ground noise, but at the cost of slower run times for simulation. Experience shows a dramatic degradation of performance with increased resolution (RNM 2007).

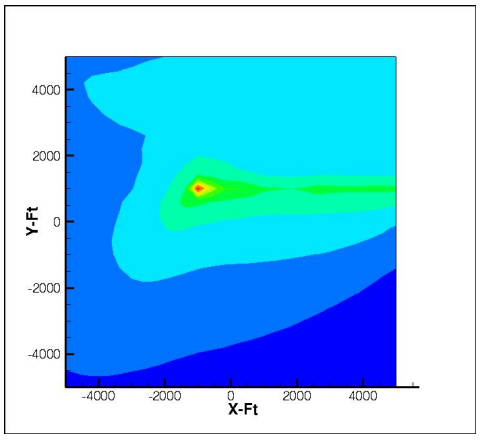


Figure 4: A Noise Contour Plot.

In (Morris et al. 2012) a method for aggregating the information in the contour plot into a scalar value to be used within noise minimization algorithms was proposed. The method is based on a *Binning Heuristic function* called Bin. Given trajectory t, and a SEL value for each grid

point (x, y), denoted $SEL(t, x, y)$, a sequence of decreasing ranges, $\langle r_1, r_2, \ldots, r_n \rangle$ partitioning the SEL values of the grid points is defined. In particular, each range corresponds to an interval of SEL values which are associated with a similar level of annoyance (FIC-Noise 1992). Moreover, given $S_i(t) = \{(x, y) | SEL(t, x, y) \in r_i\}$, the set of grid values within the range r_i, a vector $b(t) = \langle b_1(t), b_2(t), \ldots, b_n(t) \rangle$, where $b_i(t) = |S_i(t)|$, represents the number of grid points with an associate SEL value within each range. The bin-score of solution t can then be defined as $Bin(t) = \Sigma_{i=1\ldots n} w_i b_i(t)$ where w_i is the weight associated to the i-th bin, which is proportional to the annoyance level of the corresponding range and is such that $w_i > w_{i+1}$ and $\Sigma_{i=1,\ldots,n} w_i = 1$. Thus a solution that assigns lower levels of noise to larger regions of the grid is to be preferred. Weights can be tuned in various ways to penalize the presence of noisy regions in the grid.

A* and Probabilistic Road Maps

In this paper we will present a surrogate based on a machine learning technique that predicts the Bin Score value given a trajectory in input. The overall goal is to evaluate the impact of replacing the Bin scores computed from contour plots generated by RNM with scores predicted by the surrogate during optimization. The actual search will be carried out according to the well-known path planning algorithm A*. A* is a complete best-first search algorithm for discrete planning that is based on incrementally expanding a partial solution s through the use of a function $f(s)$ which estimates the minimal cost path from the start through s. $f(s)$ is the sum of the cost $g(s)$ to reach the state plus a heuristic $h(s)$ that estimates the lowest cost path to achieve the goal state that passes through s. A* enforces a best-first search strategy via a priority queue (called the open list) ordered by f-values. At each iteration, the state (node) with the lowest f-value is removed from the open list, and its neighbors generated and evaluated. The algorithm continues until a goal node has a lower value than any node in the queue. A* is complete and optimal provided the h never overestimates the cost of achieving the goal through s. In previous experiments it was shown that a good heuristic estimate for a given node is the cost of a trajectory in which altitude is maintained until the goal, but airspeed is reduced as fast as possible to the minimum velocity. This *fly high and slow* heuristic has been empirically confirmed to be admissible, but to evaluate it requires a run of RNM, which can be costly at high grid resolutions.

We also note that A* requires a way of aggregating a path-cost g as a solution is generated, where $g(x)$ is the cost from an initial state of the solution to x. Given two states, x, y, an edge distance $d(x, y)$ is defined; the aggregate cost is $g(y) = g(x) + d(x, y)$. Unfortunately, aggregating the cost for noise is problematic, because the cost is not a simple sum. It is rather, an average of the noise over time. Despite this result, following (Lindsay, Morris, and Venable 2012), we employ the cost function in which we assume factorability. In fact, although not true in general, experiments with the approximate cost function have shown that the version of A* does produce a solution that is close to optimal, and enables the sort of cost aggregation required by the algorithm (Lindsay, Morris, and Venable 2012). We refer the reader to (Lindsay, Morris, and Venable 2012) for more details on the implementation of A* in the context of rotorcraft noise minimization.

Sampling-based path planning techniques emerged out of the need to address the complexity of realistic path-planning problems. The main idea behind sampling-based approaches is to generate and organize a sequence of samples from a configuration space into a graph, where the edges are labeled with a cost. The graph is then traversed to find a path that solves the planning instance.

Probabilistic Road Maps (PRMs) (Kavraki et al. 1996) is a sampling-based method that consists of a road map construction phase and a user-defined query phase in which the roadmap is consulted for planning purposes. The basic PRM algorithm is easy to implement and performs well on a variety of problems. In (Morris et al. 2013) the authors describe how PRMs can be used to find noise-minimal landing trajectories for rotorcrafts.

The planning algorithm used in this paper is hybrid of PRM and A*. In particular first we perform a preliminary run of PRM and we find an initial path. This path is refined by an A* search performed in a more fine-grained sampled space built around the original path.

Gaussian Processes

The supervised machine learning method we investigate in this paper as a replacement model for the RNM is Gaussian Processes. Supervised machine learning (Mitchell 1997) starts with a set of correct data or experience with respect to some task and performance measure and then uses that data to infer a model that accurately approximates the real function that produced the correct data. The method of Gaussian Processes (Ebden 2008) is a robust method which provides a confidence interval on its predictions.

A Gaussian Process consists of a collection of random variables, some finite number of which have a joint Gaussian distribution (Ebden 2008). The process is fully specified by a mean function and a covariance function. The mean function, as its name implies, is simply the mean of the input data, specifically, of the predictor variable. This mean can be taken to equal zero in the absence of further information. The covariance function defines how different inputs relate to one another and gives an indication of their similarity. There are a variety of different types of covariance functions, and each is defined in terms of different parameters known as hyperparameters (Ebden 2008). Given a covariance function k, the general equation for regression in Gaussian Processes is defined as follows:

$$\hat{y^*} = K^*(\lambda^2 I + K)^{-1} y$$

where

- $\hat{y^*}$ is the model prediction, that is it is the vector of size n^* containing the predicted output for the n^* test examples.
- K^* is the cross covariance matrix that relates the covariance functions to the training data and the test data. K^* is

a $n^* \times n$ matrix where n^* and n are, respectively, the number of test and training inputs. Element $K^*_{ij} = k(x^*_i, x_j)$ where x^*_i belongs to the test set and x_j belongs to the training set.

- λ represents the noise in the training data and I is the identity matrix of size n. $\lambda^2 I$ is known as the diagonal noise matrix.
- K is the covariance matrix. K is a $n \times n$ matrix, where n is the size of the training data set. where $K_{ij} = k(x_i, x_j)$ and x_i and x_j belong to the training set.
- y is a vector of size n containing the training output data, that is the values associated by the function we want to learn to the examples contained in the training set.

We note that K^* and K are both symmetric, positive and semidefinite. A Gaussian Process is initialized by choosing values for the hyperparameters that are then optimized, during the training phase, in order to determine the optimum covariance function for the training data. In other words, a prior mean and covariance assignment based on the initial hyperparameters is specified and then a posterior mean and covariance is produced based on the prior and the data. Thus, the way the prior is specified can have significant influence on the posterior produced (Ebden 2008) . In this project we investigate the squared exponential covariance function with Automatic Relevance Determination (covSEard), defined as follows:

$$k(x_i, x_j) = \sigma^2 e^{(-(1/2)\zeta^2 |x-x|^2)}$$

where σ and ζ are the hyperparameters.

Related Work

Aside from the connections to past work in trajectory optimization and 3D path planning, recent work most relevant to that presented here is so-called surrogate modeling (or approximation modeling), usually in the context of optimal engineering design (Marsden et al. 2004). In this approach, design space exploration is performed by an inexpensive surrogate function that stands in for the real cost function. Sampling in the design parameter space using the cost function being modeled during optimization allows for the interpolation of the surrogate model. The difference between surrogate modeling and the approach here is that in our approach the surrogate is created offline using supervised machine learning in the form of Gaussian Processes.

Gaussian Process Modeling of the RNM simulator

In our setting, training and test examples are landing trajectories and the output values we want to learn is the the Bin score associated with the trajectories. Each trajectory is represented by a list of L waypoints. Each waypoint is described, in our configuration space by 3 values for its Euclidean coordinate (x, y, z) and two value for the change in velocity, $\Delta(v)$ and in altitude $\Delta(z)$. Flyability and comfort constraints have been imposed as restrictions to the values of $\Delta(v)$ and $\Delta(z)$. The training set can be represented as a $n \times 5L$ matrix where each row corresponds to a path. The training output vector y contains the n Bin score values for the training examples.

The concept underlying the training of a GP is Bayesian inference. Values for the hyperparameters of the covariance function (σ and ζ, in our case) are stored in a vector called θ. The first step is to select some initial values for θ. Then, for each choice of θ we can compute the marginal likelihood, given the training set:

$$log\ p(y|X, \theta) = (1/2)y^T K_y y - (1/2) log|K_y| - (n/2) log 2\pi$$

where $K_y = K_f + \sigma_{2n} I$ is the covariance function with noise. Training the GP model can thus be reformulated as searching for a choice of θ which maximized the marginal likelihood. This is obtained by performing a gradient descent, that is, by using partial derivatives with respect to each hyperparameter to guide the search. The complexity of this approach is dominated by that of inverting the K matrix, which is $O(n^3)$ for positive semidefinite matrices of size n. Unfortunately, the method is prone to exhibiting local minima and is highly sensitive to the initial settings θ (Ebden 2008).

Optimization with the GP surrogate

The main goal of this research is to develop a surrogate of RNM to be used during optimization in order to overcome the significant runtime overhead which RNM causes. Ideally, one would want to be able to use only the surrogate while searching for an optimal trajectory. This however was found to be not feasible due to the appearance of 'error drift' that caused by the accumulation of small error over time during search until performance of the optimizer is degraded significantly. The approach we have taken is thus that of finding a good mix of the GP model and RNM in order to get acceptable performance in terms of noise predictions in less time. The improvement should be particularly significant on problems with a high grid resolution, where the increased amount of points on which noise is evaluated by RNM takes a larger toll.

We have thus parametrized the frequency of calls to, respectively, RNM and the GP model through a 'bias' parameter with range $[0, 1]$, where a bias of 0 means only GP predictions for noise are made; and bias of 1 means only RNM is run to determine cost. Tuning this parameter allows for the determination of an optimal mixture of simulator with GP prediction that should compensate for 'error drift'.

Experimental results

We are designing a number of experiments comparing a wide range of settings that characterize both the GP model and the difficulty of the optimization problem. There are three sets of design parameters to investigate, each associated with a set of design decisions:

- Training set specification: what are the features to be trained on? What kinds of approach patterns should be used for the training? What is the size of the training set?

- GP model training: what covariance function type should be used? What initial settings of hyper-parameters should be used? How many iterations of gradient descent are required in the training?
- Planning problem algorithm and search space complexity: what optimizer planner should be used? What search and grid resolutions should be used? What bias setting should be used for the GP cost function, trading simulator with model predictions to calculate score?

Training Sets

For these experiments, the features selected for training are the sets of five tuple state vectors of a complete path (location, x, y, z, velocity v and heading h). Thus, if a path has 7 waypoints, there will be $7 \times 5 = 35$ features. Although this is a simple approach to feature selection, it has limitations that will be addressed in future work. The primary limitation is that it does not train the model on the kind of paths that are evaluated by the A* solver. Recall that A*, in general, requires to evaluate partial paths, that is paths from the start up the next decision point. In our context, the evaluation of a partial path is obtained by evaluating a completion of it in which the rotorcraft remains, from the decision point on, at a steady altitude to the goal. The GP model, by contrast, never sees such "relaxed" completions of a partial path, which may be a factor that limits the performance of the GP predictor.

Second, we experimented with training set sizes of between 50 and 500 instances. Preliminary results have not shown a large discrepancy in performance of the predictor between small and large training set sizes, although more systematic tests should be conducted. For the tests here, all models are built from training set size of 200 instances.

Finally, our system has been design to optimize paths with different patterns (straight approach, dog-leg, out-and-back, etc.). In addition, our system is currently being expanded to model obstacles in the form of restricted airspace (for example, to avoid intrusion into fixed wing landing space) as well as land use restrictions (for example, to avoid hospitals). For the experiments here, we limit the path types to a simple straight approach with no obstacles for both the training and the optimization.

GP Model

GP model training is a trial and error process for a function that is virtually unknown, such as the function define by the RNM simulator. The major effort in GP model design is in selecting the covariance function, the initial hyper-parameter settings for training, and the number of iterations in the training algorithm, which uses gradient descent to find the model that optimizes the data likelihood. Over time it was determined that the best performing covariance functions were one based on neural nets and the squared exponential (SE); the latter is the one selected for the experiments here. The SE function has a large number of hyper-parameters (specifically, D+2, where D is the number of features in the training set). An acoustic model is highly non-linear, and it is also likely that numerous local optima exist for GP covariance functions, some better than others. We found significant sensitivity in the GP model selection: minute changes in initial hyper-parameter settings often resulted in dramatic changes in the performance of the trained model on the test set, based on mean squared error between predicted and actual (RNM generated) scores. Eventually using brute force trial and error, we were able to find a GP model that worked 'pretty well'. The hyper-parameter setting that generated this model was used for all the GP models generated.

Planning Problem Specification

The primary purpose of the experiments here is to explore the tradeoffs of using a GP-based cost evaluator for path planning optimization with respect to one based on running the RNM simulator. The tradeoff is between quality and time to solve. Furthermore, we utilized the ability to modify the grid resolution of the problem to explore the tradeoffs at different resolutions. In our optimizer there are two kinds of resolution: grid resolution, discuss earlier, and what we call *search resolution*. Search resolution is a way of imposing a grid on the control decisions that expand a path during A* search, specifically on two state variables, altitude and velocity. We define a search resolution of N by M to mean that the range of possible altitude (velocity) decisions at each search point is divided into N (M) discrete choices.

For these experiments we used a single search resolution (10 by 14) and 3 different grid sizes (500, 700, and 1000). We also used 3 different settings the 'bias' parameter in the GP cost function: 0 (no runs of RNM) 1 (no GP prediction for cost evaluation) and .5 (50 percent each of RNM and GP prediction). All results are averages over 50 runs of the path planner, for which we used the pairing of PRM and A* discussed earlier in the paper.

Results and Lessons Learned

The results of the experiments are found in table 1. All scores reflect the RNM Bin value, even when GP was used during optimization, in order to be able to calculate the degradation of performance by using the GP predictor. For example, the score in row one is the result of running RNM on the path determined to be optimal by the solver using only GP prediction during search. First, we notice the insensitivity of GP time to grid resolution (by comparing the time column value for each of the rows with 0 bias value). This is to be expected, since GP prediction is sensitive only to the dimensions of the model and covariance matrix size (which is the same for each grid setting). By contrast, the time column for each of the rows with bias value 1 (only RNM simulation to evaluate paths) changes significantly as resolution is increased (from 1000 to 500). Consequently, GP modeling has higher payoff for problems where the alternative, such as running simulations, is more costly. Again, this result confirms expectations and is not surprising, although it's informative to study the rate at which these values change.

Second, note the relative degradation of performance diminishes with grid resolution growth. Thus for resolution 500 the GP-only score (Bias = 0) is 27 % worse than the RNM-only score (Bias = 1), whereas for resolution 1000 the

Search Resolution: 10 X 14				
Grid Resolution	**Bias**	**Score**	**Time**	**RNM Runs**
500	0	303.86	113.39	0
	.5	283.79	238.0	1751.84
	1.0	238.67	524.37	4781.34
700	0	165.13	113.4	0
	.5	151.9	144.2	1985.9
	1.0	125.85	299.3	4772.82
1000	0	83.22	114.38	0
	.5	74.9	112.81	2779.54
	1.0	60.53	160.1	4527.8

Table 1: Comparison of scores and run time (expressed in seconds) for GP model prediction with simulator-based prediction for optimization. Values in the right most three columns are averages over 50 runs of optimizer planner based on Probabilistic Road Maps (PRM). Three grid resolutions (500, 700, 1000) and three bias settings (0, .5 and 1) are shown, where 0 bias means only GP predictions are made (i.e. 0 RNM runs) and a bias of 1 means no GP predictions are made. The GP models were trained on data sets of size 200. The table illustrates the trade off between time and quality by using a GP model during optimization.

GP-only score is 37 % worse. Thus, again as expected, GP-only is a better alternative as resolution (complexity of the problem) is increased. The ability to tune the bias offers designers a range of options for trading score with speed.

For planning problems similar to the one discussed in this paper, the results of these experiments suggest that machine learning has the potential to offer useful approximations to complex engineering systems for evaluating the cost of design decisions, specifically, the design of paths using planners like A* and PRMs. The cost of using machine learning is mostly incurred 'off line', first in the generation of training sets for GP modeling (which of course can take time but is fully automated), and second in the design and training of a GP predictor, which potentially includes many hours of human trial and error as well as machine cycles. We plan to expend more (human and machine) effort in the design of GP models to try to come closer to the RNM-only performance.

Summary and Future Work

Future work will investigate and document experiments over a wider range of experimental designs specifically geared towards assessing the interdependence of approximation quality and solution quality. In first instance, we will analyze how the training set size affects the performance of the predictor. We are also eager to explore one advantage of GPs that were not exploited in the experiments here, namely that GP prediction returns both a mean value and a variance measure. Variance can be used as an indicator of the confidence in the prediction made; higher variance means less confidence. We're considering a smarter alternative to the bias parameter discussed above, in which variance determines whether the simulator needs to be run to determine cost.

We have also started modeling land usage and restricted airspace conditions. We not how our planning approaches as well as the Bin scoring rule allow for a natural embedding of such features by means of obstacles and reweighing of grid portions. In the long term, we also plan to extend our work to multi-rotorcraft scenarios by exploiting a related simulation capability already available in RNM.

In this paper we have considered the design of a surrogate for a computationally intensive rotorcraft noise simulation tool based on Gaussian processes. We have tested the utility of the surrogate in the context of searching for noise-minimal landing trajectories. The experimental results suggest the high potential of the surrogate both in terms of efficiency, accuracy and robustness to different grid resolutions. In future work we will conduct additional experiments on the use of the surrogate coupled with different search algorithms, such as, local search and PRM.

References

Ebden, M. 2008. *Gaussian Processes for Regression: a quick Introduction*. University of Oxford.

FIC-Noise. 1992. 1992 federal interagency commitee on noise (ficon) report - federal agency review of selected airport noise analysis issues. Technical report, Federal Interagency Committee on Noise.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on robotics an automation, Vol.12, No.4*.

Lindsay, J.; Morris, R.; and Venable, K. 2012. Automated design of noise-minimal, safe rotorcraft trajectories. In *Proceedings of the 68th American Helicopter Society Annual Forum & Technology Display, Acoustics Session.* AHS International.

Marsden, A. A.; Wang, M.; Jr., J. E. D.; and Moin, P. 2004. Optimal aeroacoustic shape design using the surrogate management framework. *Optimization and Engineering* 5.

Mitchell, T. 1997. *Machine Learning*. McGraw Hill.

Morris, R.; Venable, K. B.; Pegoraro, M.; and Lindsay, J. 2012. Local search for designing noise-minimal rotorcraft approach trajectories. In *IAAI*. AAAI.

Morris, R. A.; Donini, M.; Venable, K.; and Johnson, M. 2013. Designing quiet rotorcraft landing trajectories with probabilistic road maps. In *Workshops on Scheduling and Planning Applications woRKshop (SPARK 2013)*.

RNM. 2007. *Rotorcraft Noise Model Technical Reference and User Manual (Version 7)*. NASA Langley Research Center.