
ICAPS05

WS6

# Workshop on Planning under Uncertainty for Autonomous Systems

Christophe Guettier
*SAGEM SA, FRANCE*

Neil Yorke-Smith
*SRI International, USA*

**ICAPS 2005**
**Monterey, California, USA**
**June 6-10, 2005**

**CONFERENCE CO-CHAIRS:**

Susanne Biundo
*University of Ulm, GERMANY*

Karen Myers
*SRI International, USA*

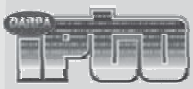
Kanna Rajan
*NASA Ames Research Center, USA*

**Cover design: L.Castillo@decsai.ugr.es**

# Workshop on Planning under Uncertainty for Autonomous Systems

Christophe Guettier
*SAGEM SA, FRANCE*

Neil Yorke-Smith
*SRI International, USA*



# Workshop on Planning under Uncertainty for Autonomous Systems

## Table of contents


# Workshop on Planning under Uncertainty for Autonomous Systems

## Preface

*Autonomous systems UAVs, UUVs, UGVs, planetary rovers and space probes among them operate with limited human intervention in changing and partially known environments. These systems must plan and control internal behaviour while responding to mission updates and environment changes. The promise of greater mission capability at lower cost is leading to increasing interest in autonomy across a spectrum of scientific, commercial and military application domains. The papers accepted for this workshop form a representative sample of real-world autonomous systems, operating in air, sea, space and ground environments.*

*Since simple heuristic approaches dramatically reduce the operational scope of autonomous systems, planning problems appear to be system centric and critical for the success of challenging missions. Various theoretical and practical aspects of planning for such systems are considered in the workshop. Extensions to classical modelling techniques are explored, based on constraint programming or probability representations. Combining uncertainty and resource management highlights the borderline between stochastic and non-stochastic modelling approaches. Multiple approaches to algorithms are also investigated to target search problems relevant to autonomous systems, such as path planning in navigation, and planning the mode of operations for equipment and on-board systems management. These works provide a strong basis to architect systems which deliver good plans just in time, finding the right balance between off-line and reactive methods. Lastly, managing multiple systems with some degree of autonomy constellations of spacecraft and swarms of UAVs introduces uncertainty, biased and incomplete data. Papers in the workshop provide perspectives to tackle part of these problems at planning time.*

*In addition to the seven accepted technical papers, the workshop features two short position papers, and the invited talk Intelligent Mission Planning and Control of Autonomous Underwater Vehicles by Roy Turner from the Maine Software Agents and Artificial Intelligence Laboratory, University of Maine.*

*We would like to thank members of the Programme Committee who provided excellent, timely, and constructive reviews. Special thanks are due to Gerard Verfaillie for managing the commentary process. We are also grateful to the ICAPS conference and workshop chairs for their help in organizing this workshop.*

*Organizers*

- *Christophe Guettier (SAFRAN, France)*
- *Neil Yorke-Smith (SRI, USA)*

*Programme Committee*

- *Christopher Beck (University of Toronto, Canada)*
- *Patrick Fabiani (ONERA, France)*
- *Karen Haigh (Honeywell, USA)*
- *Detlef Koschny (ESA ESTEC, Netherland)*
- *Juliette Mattioli (Thales, France)*
- *Nicola Muscetolla (NASA AMES, USA)*
- *Yi Shang (University of Missouri-Columbia, USA)*
- *Roy Turner (University of Maine, USA)*
- *Gerard Verfaillie (ONERA, France)*
- *Wheeler Ruml (PARC, USA)*

# Intelligent Mission Planning and Control of Autonomous Underwater Vehicles

**Roy M. Turner**
Maine Software Agents and Artificial Intelligence Laboratory
Department of Computer Science
University of Maine
Orono, ME 04469
rmt@umcs.maine.edu

**Abstract**

Autonomous underwater vehicles (AUVs) undertaking complex missions require plans to carry out those missions. In an ideal world, this planning could be done off-line, with the AUV being given the plan to execute in the water. The real world is not this nice, of course. Knowledge about the underwater environment is uncertain and incomplete. The world is dynamic, with processes and other agents that change the world in unpredictable ways. Sensors for AUVs are notoriously imprecise and noisy, and the AUV, being a real physical system, often responds to commands in imprecise, unpredictable ways. The resulting uncertainty can cause off-line plans to fail at execution time as assumptions upon which they were based are violated. Consequently, AUVs need on-board planners with the ability to modify or replan and, in many cases, to create new plans.

This paper and the accompanying talk discuss the sources and results of uncertainty in AUV mission control. The talk also provides an overview of past and current planning technologies used in the AUV domain and a discussion of future directions.

Autonomous underwater vehicles (AUVs) are unmanned, untethered submersibles. AUVs have great promise for a variety of purposes, both civilian and military. For example, AUVs have been used or proposed for use for ocean science applications, global change and other environmental monitoring, oil spill and other pollution remediation, ground-truthing satellite date, underwater construction and inspection, cable laying, aquaculture, military missions, and industrial applications.

AUV hardware is mature enough for many of these purposes. Today, there is a wide variety of vehicles, with differing ranges, depth capabilities, sensors, and effectors. Vehicles range from very low-cost vehicles with limited duration and depths, such as the Virginia Tech Miniature AUV (Stilwell *et al.* 2004), to very capable, medium-range scientific vehicles, such as Florida Atlantic University's Ocean Explorer and Ocean Odyssey vehicles (e.g., Smith *et al.* 1996; see Fig. 1) and the Naval Postgraduate University's Phoenix (Brutzman *et al.* 1998) and ARIES vehicles (Fig. 2), to long-range vehicles such as the Autonomous Undersea Systems Institute's solar-powered AUVs (Jalbert *et al.* 1997; see Fig. 3) and various versions of the Slocum gliders (Bachmayer *et al.* 2004), to full-ocean depth, long-duration vehicles such as Woods Hole's ABE (Yoerger, Bradley, & Walden 1991). Military AUVs run the gamut from experimental scientific vehicles (e.g., ARIES) to sophisticated sensor platforms and intelligent torpedoes. While some AUVs require substantial support, including vessels and trained support staff, others are fieldable by graduate students operating from shore or from a small boat.



Figure 1: Florida Atlantic University's Ocean Explorer.

Vehicle software has lagged behind hardware, however. While low-level controllers, based on control theory, are quire competent and robust, as are path-planning algorithms, overall high-level mission control is generally done by relatively simple, inflexible software. Some AUVs are controlled by subsumption-based controllers (e.g., Bellingham *et al.* 1994) or controllers based on finite state machines (e.g., Phoha, Peluso, & Culver 2001), and so are not particularly compatible with on-board planning. For others, if automated mission planning is done at all, it is almost always done off-line, with the plans downloaded to the AUV's mission controller for execution (e.g., (Brutzman *et al.* 1998)).

Part of the reason for this is the typical origin of AUVs in mechanical engineering or ocean engineering rather than computer science. Part, too, is due to the difficulty of the do-


Figure 2: Naval Postgraduate School's ARIES AUV.

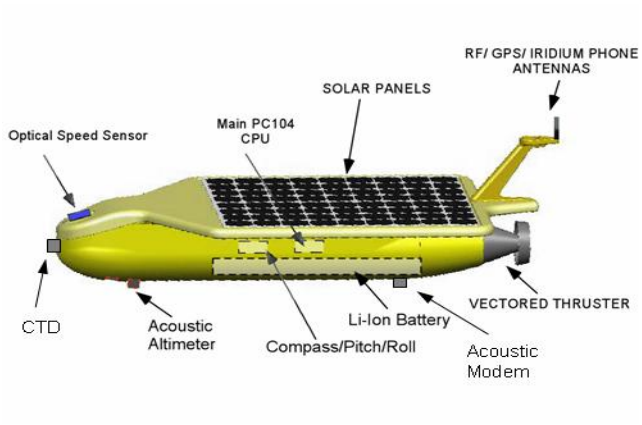



Figure 3: The Autonomous Undersea Systems Institute's Solar AUV.

main in terms of the high degree of uncertainty and incomplete knowledge, its dynamic nature, and its complexity—given the state of the art in planning technology, most planners are simply not up to the task. Part, too, is due to the needs of the users of the AUVs. Scientists want assurance that data is collected where they specify, when they specify, not at the whim of an AUV controller with too much autonomy. The military, too, has often been opposed to on-board planners, since it is critical for most of their missions that an AUV's behavior be predictable: one does not want a weapon, for example, with too much of a mind of its own.[1] And, finally, part is due to the fact that current non-planning control software is adequate to the rather simple uses to which AUVs have so far been put.

For more advanced missions, however, this will have to change. Missions in which the AUV must exhibit a high degree of autonomy, long-duration missions, missions in highly dynamic environments, and complex missions involving multiple vehicles all will require capabilities far beyond the current state of the art in AUV control. In particular, such missions will require AUVs that are capable of replanning or repairing downloaded missions or of planning their own missions in the first place.

Here, I will briefly discuss some causes of the uncertainty that, in large part, makes planning in the AUV domain difficult. In the accompanying talk, I will survey some of the past and current approaches to planning in the AUV domain, discuss the state of the art, and conclude with a look at what the future may hold for planner-based AUV mission controllers.

The most basic reason to use planning technology for AUVs is, of course, the same as for any other agent: to correctly sequence actions to accomplish users' goals. If there is no uncertainty involved, then off-line planning is sufficient. This means that the agent must be operating a well-known, static environment, and it must have certain knowledge about the environment and itself as well as accurate sensors.

If there is uncertainty, however, then the AUV needs to be able at least to modify its plan or even replan completely. Uncertainty undermines the assumptions upon which the off-line plan was based. For example, if the AUV is following a plan to retrieve the black box from a downed aircraft from location $x$, but the AUV's knowledge is imprecise so that the black box could be anywhere in an area of radius $\epsilon$ around that point, then there is a good chance that AUV will not be able to find the target simply by following its plan.

An AUV will encounter uncertainty to one degree or another on almost any mission. This is due to a variety of factors in the AUV's environment, the mission, and itself. One factor is the inherent incompleteness of the AUV's knowledge about its environment. In many respects, relatively little is known about the ocean, especially the deep ocean. Indeed, it was noted some time ago that we know less about Neptune's realm than we do about the planet Neptune (Blidberg, Turner, & Chappell 1991); this is still true to a large extent. Consequently, predications about the environment upon which any *a priori* plan is based will often be violated during the plan's execution.

A dynamic environment can also lead to uncertainty. If the world were completely deterministic, complete knowledge of the world would allow predictions to be made with complete certainty about how it will change. However, this is not the case. In general, stochastic processes, as well as uncertain or incomplete knowledge of processes operating in the world, will lead to changes occurring that a planner cannot predicate ahead of time. This, too, can undermine the plan at execution time.

Imprecision in the AUV's sensors also contributes to uncertainty. For example, sonar is notoriously undependable, especially underwater, being susceptible to such things a bouncing off thermoclines or other density changes in the water. Even localization is uncertain. Except in very shallow water, AUVs cannot make use of GPS on a regular basis to determine their position. Unless it is feasible to have transponders placed for long-baseline navigation, they have to rely on such things as inertial guidance or dead reckoning

[1] For a humorous treatment of this idea, see the 1974 movie "Dark Star".

to estimate their position between GPS fixes. Even cameras suffer problems in some underwater environments. For example, in the temperate and boreal ocean, suspended particulate matter severely reduces visibility. This also means that for all practical purposes, in these environments computer vision is not viable for an AUV, thus depriving it of a valuable sensory modality and consequently increasing its uncertainty about the world around it.

The AUV's own state and the effects of its actions may be uncertain, as well. For example, fault indicators may themselves be faulty, battery charge indicators may be wrong, and so forth. In addition, effectors are mechanical devices that may not perform as expected: robot arms may miss their targets, thrusters may not turn propellers the number of revolutions expected, the entire AUV may fail, etc. Consequently, the AUV may not be able to predict with complete accuracy its own state (or trends that predict its future state) or the results of its own actions.

The likelihood and severity of uncertainty is increased by long-duration or multiagent missions. Not only does uncertainty accumulate in such things as position estimates, but the effects of uncertainty on the plan also accumulate. Multiagent missions increase uncertainty by the actions of the often unpredictable agents that are part of the MAS. In addition, in some "open" multiagent systems, such as some visions for autonomous oceanographic sampling networks (AOSNs; e.g., Turner & Turner 2001), which AUVs are present may change over time as vehicles exit (e.g., due to failure or for maintenance) or enter the system. This increases the dynamic nature of the environment and, hence, the AUV's uncertainty about the world.

There are planning techniques that can deal with some uncertainty in the environment, of course, such as conditional planners (see, e.g., Russell & Norvig 2003), including the extreme case of so-called universal plans (e.g., Schoppers 1987), as can some non-planning techniques, such as reactive planning (e.g., Georgeff & Lansky 1987) and the subsumption architecture (Brooks 1986; Bellingham *et al.* 1994). However, uncertainty, if severe, can still be catastrophic to *a priori* plans, and the latter approaches are not able easily to sequence actions in service of goals.

In some circumstances, the AUV has no choice but to create its plan in the field. This could happen if the AUV was re-tasked while on-site. This is likely for long-duration missions, such as missions in which the AUV is acting in the capacity of an "underwater satellite" that remains on station for long periods of time to return data. It could also happen if the AUV is part of a multi-AUV system (i.e., a multiagent system, or MAS), such as an autonomous oceanographic sampling network (Curtin *et al.* 1993). For some MAS control approaches, agents are assigned tasks on the fly by other agents or bid for tasks and must create plans to accomplish them (e.g., Smith 1980). In still other cases, offline planning may be impossible for the mission goal due to a lack of *a priori* knowledge. For example, a goal such as "photograph any interesting objects in an area", even where "interesting" is well-defined, is impossible to plan for fully before the objects are seen.

Other challenges for planners in the AUV domain abound. For example, even for non-covert missions, communication with AUVs is extremely limited. Unless the AUV breaks the surface, in the ocean the only practical way to communicate over any significant distance is via acoustic modem. This can require a great deal of the AUV's limited power, and the bandwidth is quite limited: from around 20 Kbps in the open ocean to 20 bps in the surf zone. One implication of this is that an AUV mission controller for many missions needs to be capable of autonomy. Another implication for multiagent missions is that the organizational structure and coordination mechanism for MAS must be selected so as to tread lightly on the communication channel.

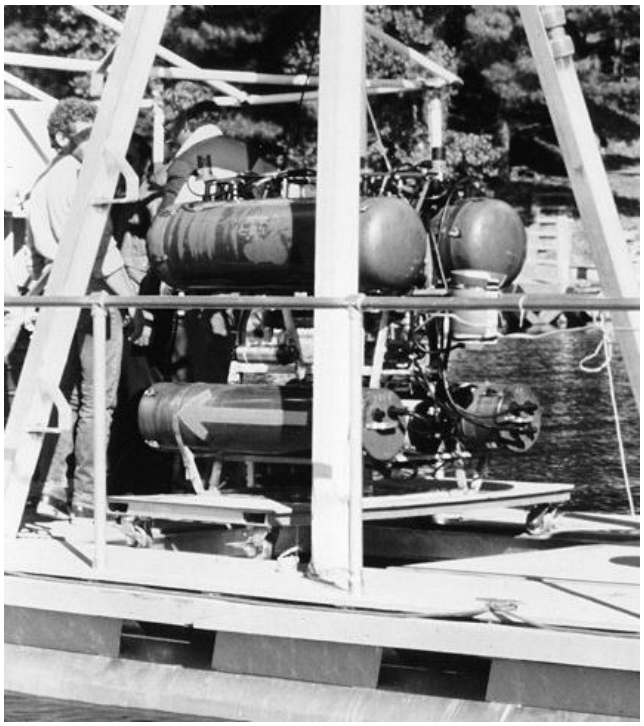


Figure 4: AUSI's EAVE (Experimental Autonomous underwater VEhicle).

The accompanying talk will discuss approaches to planning for AUVs, starting with very early work on the EAVE architecture (Blidberg & Chappell 1986; see Fig. 4), through current off-line approaches (e.g., for ARIES), and continuing with emerging approaches to on-board planners (including our own Orca intelligent mission controller [Turner 1995; Turner 1998]). The talk will discuss both single-agent and multiagent systems, and it will also discuss future directions for planning in the AUV domain.

## References

Bachmayer, R.; Rice, J.; Creber, R.; and Fletcher, C. 2004. Navigation and control of multiple gliders in an underwater acoustic network. In *IEEE/OES Autonomous Underwater Vehicles 2004 (AUV'04): A Workshop on Multiple AUV Operations*. Sebasco Estates, Maine: IEEE.

Bellingham, J.; Goudey, C.; Consi, T.; Bales, J.; and Atwood, D. 1994. A second-generation survey AUV. In *Proceedings of the 1994 IEEE Symposium on Autonomous Underwater Vehicle Technology (AUV'94)*, 148–155.

Blidberg, D. R., and Chappell, S. G. 1986. Guidance and control architecture for the EAVE vehicle. *IEEE Journal of Oceanic Engineering* OE–11(4):449–461.

Blidberg, D. R.; Turner, R. M.; and Chappell, S. G. 1991. Autonomous underwater vehicles: Current activities and research opportunities. *Robotics and Autonomous Systems* 7:139–150.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA–2(1):14–23.

Brutzman, D.; Healey, T.; Marco, D.; and McGhee, B. 1998. The Phoenix autonomous underwater vehicle. In Kortenkamp, D.; Bonasso, P.; and Murphy, R., eds., *AI-Based Mobile Robots*. MIT/AAAI Press. chapter 13.

Curtin, T.; Bellingham, J.; Catipovic, J.; and Webb, D. 1993. Autonomous oceanographic sampling networks. *Oceanography* 6(3).

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning: An experiment with a mobile robot. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 677–682.

Jalbert, J. C.; Irazoqui-Pastor, P.; Miles, S.; Blidberg, D. R.; and James, D. 1997. Solar AUV technology evaluation and development project. In *Proceedings of the Tenth International Symposium on Unmanned Untethered Submersible Technology*, 75–87.

Phoha, S.; Peluso, E.; and Culver, R. 2001. A high-fidelity ocean sampling mobile network (SAMON) simulator testbed for evaluating intelligent control of unmanned underwater vehicles. *IEEE Journal of Oceanic Engineering* 26(4):646–653.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 2nd edition.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1039–1046.

Smith, S.; Ganesan, K.; Dunn, S.; and An, P. 1996. Strategies for simultaneous multiple AUV operation and control. In *IARP'96*.

Smith, R. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* C–29(12):1104–1113.

Stilwell, D. J.; Gadre, A. S.; Sylvester, C. A.; and Cannell, C. J. 2004. Design elements of a small low-cost autonomous underwater vehicle for field experiments in multi-vehicle coordination. In *IEEE/OES Autonomous Underwater Vehicles 2004 (AUV'04): A Workshop on Multiple AUV Operations*. Sebasco Estates, Maine: IEEE.

Turner, R. M., and Turner, E. H. 2001. A two-level, protocol-based approach to controlling autonomous oceanographic sampling networks. *IEEE Journal of Oceanic Engineering* 26(4).

Turner, R. M. 1995. Intelligent control of autonomous underwater vehicles: The Orca project. In *Proceedings of the 1995 IEEE International Conference on Systems, Man, and Cybernetics*. Vancouver, Canada.

Turner, R. M. 1998. Context-mediated behavior for intelligent agents. *International Journal of Human–Computer Studies* 48(3):307–330.

Yoerger, D. R.; Bradley, A. M.; and Walden, B. B. 1991. The autonomous benthic explorer (ABE): An AUV optimized for deep seafloor studies. In *Proceedings of the Seventh International Symposium on Unmanned Untethered Submsersible Technology*, 60–70. Durham, New Hampshire: Marine Systems Engineering Laboratory, University of New Hampshire.

# A Guide to Heuristic-based Path Planning

**Dave Ferguson, Maxim Likhachev, and Anthony Stentz**
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA

**Abstract**

We describe a family of recently developed heuristic-based algorithms used for path planning in the real world. We discuss the fundamental similarities between static algorithms (e.g. A*), replanning algorithms (e.g. D*), anytime algorithms (e.g. ARA*), and anytime replanning algorithms (e.g. AD*). We introduce the motivation behind each class of algorithms, discuss their use on real robotic systems, and highlight their practical benefits and disadvantages.

## Introduction

In this paper, we describe a family of heuristic-based planning algorithms that has been developed to address various challenges associated with planning in the real world. Each of the algorithms presented have been verified on real systems operating in real domains. However, a prerequisite for the successful general use of such algorithms is (1) an analysis of the common fundamental elements of such algorithms, (2) a discussion of their strengths and weaknesses, and (3) guidelines for when to choose a particular algorithm over others. Although these algorithms have been documented and described individually, a comparative analysis of these algorithms is lacking in the literature. With this paper we hope to fill this gap.

We begin by providing background on path planning in static, known environments and classical algorithms used to generate plans in this domain. We go on to look at how these algorithms can be extended to efficiently cope with partially-known or dynamic environments. We then introduce variants of these algorithms that can produce suboptimal solutions very quickly when time is limited and improve these solutions while time permits. Finally, we discuss an algorithm that combines principles from all of the algorithms previously discussed; this algorithm can plan in dynamic environments *and* with limited deliberation time. For all the algorithms discussed in this paper, we provide example problem scenarios in which they are very effective and situations in which they are less effective. Although our primary focus is on path planning, several of these algorithms are applicable in more general planning scenarios.



Our aim is to share intuition and lessons learned over the course of several system implementations and guide readers in choosing algorithms for their own planning domains.

## Path Planning

Planning consists of finding a sequence of actions that transforms some initial state into some desired goal state. In path planning, the states are agent locations and transitions between states represent actions the agent can take, each of which has an associated cost. A path is *optimal* if the sum of its transition costs (edge costs) is minimal across all possible paths leading from the initial position (start state) to the goal position (goal state). A planning algorithm is *complete* if it will always find a path in finite time when one exists, and will let us know in finite time if no path exists. Similarly, a planning algorithm is optimal if it will always find an optimal path.

Several approaches exist for computing paths given some representation of the environment. In general, the two most popular techniques are deterministic, heuristic-based algorithms (Hart, Nilsson, & Rafael 1968; Nilsson 1980) and randomized algorithms (Kavraki *et al.* 1996; LaValle 1998; LaValle & Kuffner 1999; 2001).

When the dimensionality of the planning problem is low, for example when the agent has only a few degrees of freedom, deterministic algorithms are usually favored because they provide bounds on the quality of the solution path returned. In this paper, we concentrate on deterministic algorithms. For more details on probabilistic techniques, see (LaValle 2005).

A common technique for robotic path planning consists of representing the environment (or configuration space) of the robot as a graph $G = (S, E)$, where $S$ is the set of possible robot locations and $E$ is a set of edges that represent transitions between these locations. The cost of each edge represents the cost of transitioning between the two endpoint locations.

Planning a path for navigation can then be cast as a search problem on this graph. A number of classical graph search algorithms have been developed for calculating least-cost paths on a weighted graph; two popular ones are Dijkstra's algorithm (Dijkstra 1959) and A* (Hart, Nilsson, & Rafael 1968; Nilsson 1980). Both algorithms return an optimal path (Gelperin 1977), and can be considered as special forms of

**ComputeShortestPath**()
01. while (argmin$_{s \in OPEN}(g(s) + h(s, s_{goal})) \neq s_{goal}$)
02. remove state $s$ from the front of *OPEN*;
03. for all $s' \in Succ(s)$
04. if $(g(s') > g(s) + c(s, s'))$
05. $g(s') = g(s) + c(s, s')$;
06. insert $s'$ into *OPEN* with value $(g(s') + h(s', s_{goal}))$;

**Main**()
07. for all $s \in S$
08. $g(s) = \infty$;
09. $g(s_{start}) = 0$;
10. $OPEN = \emptyset$;
11. insert $s_{start}$ into *OPEN* with value $(g(s_{start}) + h(s_{start}, s_{goal}))$;
12. ComputeShortestPath();

Figure 1: **The A* Algorithm (forwards version).**

dynamic programming (Bellman 1957). A* operates essentially the same as Dijkstra's algorithm except that it guides its search towards the most promising states, potentially saving a significant amount of computation.

A* plans a path from an initial state $s_{start} \in S$ to a goal state $s_{goal} \in S$, where $S$ is the set of states in some finite state space. To do this, it stores an estimate $g(s)$ of the path cost from the initial state to each state $s$. Initially, $g(s) = \infty$ for all states $s \in S$. The algorithm begins by updating the path cost of the start state to be zero, then places this state onto a priority queue known as the *OPEN* list. Each element $s$ in this queue is ordered according to the sum of its current path cost from the start, $g(s)$, and a heuristic estimate of its path cost to the goal, $h(s, s_{goal})$. The state with the minimum such sum is at the front of the priority queue. The heuristic $h(s, s_{goal})$ typically underestimates the cost of the optimal path from $s$ to $s_{goal}$ and is used to focus the search.

The algorithm then pops the state $s$ at the front of the queue and updates the cost of all states reachable from this state through a direct edge: if the cost of state $s$, $g(s)$, plus the cost of the edge between $s$ and a neighboring state $s'$, $c(s, s')$, is less than the current cost of state $s'$, then the cost of $s'$ is set to this new, lower value. If the cost of a neighboring state $s'$ changes, it is placed on the *OPEN* list. The algorithm continues popping states off the queue until it pops off the goal state. At this stage, if the heuristic is *admissible*, i.e. guaranteed to not overestimate the path cost from any state to the goal, then the path cost of $s_{goal}$ is guaranteed to be optimal. The complete algorithm is given in Figure 1.

It is also possible to switch the direction of the search in A*, so that planning is performed from the goal state towards the start state. This is referred to as 'backwards' A*, and will be relevant for some of the algorithms discussed in the following sections.

## Incremental Replanning Algorithms

The above approaches work well for planning an initial path through a known graph or planning space. However, when operating in real world scenarios, agents typically do not have perfect information. Rather, they may be equipped with incomplete or inaccurate planning graphs. In such cases, any path generated using the agent's initial graph may turn out to be invalid or suboptimal as it receives updated information. For example, in robotics the agent may be equipped with an onboard sensor that provides updated environment information as the agent moves. It is thus important that the agent is able to update its graph and replan new paths when new information arrives.


Pioneers Automated E-Gator

Figure 2: D* and its variants are currently used for path planning on several robotic systems, including indoor planar robots (Pioneers) and outdoor robots operating in more challenging terrain (E-Gators).

One approach for performing this replanning is simply to replan from scratch: given the updated graph, a new optimal path can be planned from the robot position to the goal using A*, exactly as described above. However, replanning from scratch every time the graph changes can be very computationally expensive. For instance, imagine that a change occurs in the graph that does not affect the optimality of the current solution path. Or, suppose some change takes place that does affect the current solution, but in a minor way that can be quickly fixed. Replanning from scratch in either of these situations seems like a waste of computation. Instead, it may be far more efficient to take the previous solution and repair it to account for the changes to the graph.

A number of algorithms exist for performing this repair (Stentz 1994; 1995; Barbehenn & Hutchinson 1995; Ramalingam & Reps 1996; Ersson & Hu 2001; Huiming *et al.* 2001; Podsedkowski *et al.* 2001; Koenig & Likhachev 2002). Focussed Dynamic A* (D*) (Stentz 1995) and D* Lite (Koenig & Likhachev 2002) are currently the most widely used of these algorithms, due to their efficient use of heuristics and incremental updates. They have been used for path planning on a large assortment of robotic systems, including both indoor and outdoor platforms (Stentz & Hebert 1995; Hebert, McLachlan, & Chang 1999; Matthies *et al.* 2000; Thayer *et al.* 2000; Zlot *et al.* 2002; Likhachev 2003) (see Figure 2). They have also been extended to provide incremental replanning behavior in symbolic planning domains (Koenig, Furcy, & Bauer 2002).

D* and D* Lite are extensions of A* able to cope with changes to the graph used for planning. The two algorithms are fundamentally very similar; we restrict our attention here to D* Lite because it is simpler and has been found to be slightly more efficient for some navigation tasks (Koenig & Likhachev 2002). D* Lite initially constructs an optimal solution path from the initial state to the goal state in exactly the same manner as backwards A*. When changes to the

planning graph are made (i.e., the cost of some edge is altered), the states whose paths to the goal are immediately affected by these changes have their path costs updated and are placed on the planning queue (*OPEN* list) to propagate the effects of these changes to the rest of the state space. In this way, only the affected portion of the state space is processed when changes occur. Furthermore, D* Lite uses a heuristic to further limit the states processed to only those states whose change in path cost could have a bearing on the path cost of the initial state. As a result, it can be up to two orders of magnitude more efficient than planning from scratch using A* (Koenig & Likhachev 2002).

In more detail, D* Lite maintains a least-cost path from a start state $s_{start} \in S$ to a goal state $s_{goal} \in S$, where $S$ is again the set of states in some finite state space. To do this, it stores an estimate $g(s)$ of the cost from each state $s$ to the goal. It also stores a one-step lookahead cost $rhs(s)$ which satisfies:

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{goal} \\ \min_{s' \in Succ(s)}(c(s,s') + g(s')) & \text{otherwise,} \end{cases}$$

where $Succ(s) \in S$ denotes the set of successors of $s$ and $c(s,s')$ denotes the cost of moving from $s$ to $s'$ (the edge cost). A state is called consistent iff its g-value equals its rhs-value, otherwise it is either overconsistent (if $g(s) > rhs(s)$) or underconsistent (if $g(s) < rhs(s)$).

Like A*, D* Lite uses a heuristic and a priority queue to focus its search and to order its cost updates efficiently. The heuristic $h(s,s')$ estimates the cost of moving from state $s$ to $s'$, and needs to be admissible and (backward) consistent: $h(s,s') \leq c^*(s,s')$ and $h(s,s'') \leq h(s,s') + c^*(s',s'')$ for all states $s, s', s'' \in S$, where $c^*(s,s')$ is the cost associated with a least-cost path from $s$ to $s'$. The priority queue *OPEN* always holds exactly the inconsistent states; these are the states that need to be updated and made consistent.

The priority, or *key value*, of a state $s$ in the queue is:

$$\begin{aligned} key(s) &= [k_1(s), k_2(s)] \\ &= [\min(g(s), rhs(s)) + h(s_{start}, s), \\ &\quad \min(g(s), rhs(s))]. \end{aligned}$$

A lexicographic ordering is used on the priorities, so that priority $key(s)$ is less than or equal to priority $key(s')$, denoted $key(s) \dot{\leq} key(s')$, iff $k_1(s) < k_1(s')$ or both $k_1(s) = k_1(s')$ and $k_2(s) \leq k_2(s')$. D* Lite expands states from the queue in increasing priority, updating their g-values and their predecessors' rhs-values, until there is no state in the queue with a priority less than that of the start state. Thus, during its generation of an initial solution path, it performs in exactly the same manner as a backwards A* search.

To allow for the possibility that the start state may change over time D* Lite searches backwards and consequently focusses its search towards the start state rather than the goal state. If the g-value of each state $s$ was based on a least-cost path from $s_{start}$ to $s$ (as in forward search) rather than from $s$ to $s_{goal}$, then when the robot moved every state would have to have its cost updated. Instead, with D* Lite only the heuristic value associated with each inconsistent state needs to be updated when the robot moves. Further, even this step can be avoided by adding a bias to newly inconsistent states being added to the queue (see (Stentz 1995) for details).

**key**$(s)$
01. return $[min(g(s), rhs(s)) + h(s_{start}, s); min(g(s), rhs(s)))]$;

**UpdateState**$(s)$
02. if $s$ was not visited before
03. $\quad g(s) = \infty$;
04. if $(s \neq s_{goal})$ $rhs(s) = min_{s' \in Succ(s)}(c(s,s') + g(s'))$;
05. if ($s \in$ *OPEN*) remove $s$ from *OPEN*;
06. if $(g(s) \neq rhs(s))$ insert $s$ into *OPEN* with key($s$);

**ComputeShortestPath**()
07. while ($\min_{s \in OPEN}$(key($s$)) $\dot{<}$ key($s_{start}$) OR $rhs(s_{start}) \neq g(s_{start})$)
08. $\quad$ remove state $s$ with the minimum key from *OPEN*;
09. $\quad$ if $(g(s) > rhs(s))$
10. $\quad\quad g(s) = rhs(s)$;
11. $\quad\quad$ for all $s' \in Pred(s)$ UpdateState($s'$);
12. $\quad$ else
13. $\quad\quad g(s) = \infty$;
14. $\quad\quad$ for all $s' \in Pred(s) \cup \{s\}$ UpdateState($s'$);

**Main**()
15. $g(s_{start}) = rhs(s_{start}) = \infty$; $g(s_{goal}) = \infty$;
16. $rhs(s_{goal}) = 0$; *OPEN* $= \emptyset$;
17. insert $s_{goal}$ into *OPEN* with key($s_{goal}$);
18. forever
19. $\quad$ ComputeShortestPath();
20. $\quad$ Wait for changes in edge costs;
21. $\quad$ for all directed edges $(u, v)$ with changed edge costs
22. $\quad\quad$ Update the edge cost $c(u, v)$;
23. $\quad\quad$ UpdateState($u$);

Figure 3: **The D* Lite Algorithm (basic version).**

When edge costs change, D* Lite updates the rhs-values of each state immediately affected by the changed edge costs and places those states that have been made inconsistent onto the queue. As before, it then expands the states on the queue in order of increasing priority until there is no state in the queue with a priority less than that of the start state. By incorporating the value $k_2(s)$ into the priority for state $s$, D* Lite ensures that states that are along the current path and on the queue are processed in the right order. Combined with the termination condition, this ordering also ensures that a least-cost path will have been found from the start state to the goal state when processing is finished. The basic version of the algorithm (for a fixed start state) is given in Figure 3[1].

D* Lite is efficient because it uses a heuristic to restrict attention to only those states that could possibly be relevant to repairing the current solution path from a given start state to the goal state. When edge costs decrease, the incorporation of the heuristic in the key value ($k_1$) ensures that only those newly-overconsistent states that could potentially decrease the cost of the start state are processed. When edge costs increase, it ensures that only those newly-underconsistent states that could potentially invalidate the current cost of the start state are processed.

In some situations the process of invalidating old costs

[1]Because the optimizations of D* Lite presented in (Koenig & Likhachev 2002) can significantly speed up the algorithm, for an efficient implementation of D* Lite please refer to that paper.

may be unnecessary for repairing a least-cost path. For example, such is the case when there are no edge cost decreases and all edge cost increases happen outside of the current least-cost path. To guarantee optimality in the future, D* Lite would still invalidate portions of the old search tree that are affected by the observed edge cost changes even though it is clear that the old solution remains optimal. To overcome this a modified version of D* Lite has recently been proposed that delays the propagation of cost increases as long as possible while still guaranteeing optimality. Delayed D* (Ferguson & Stentz 2005) is an algorithm that initially ignores underconsistent states when changes to edge costs occur. Then, after the new values of the overconsistent states have been adequately propagated through the state space, the resulting solution path is checked for any underconsistent states. All underconsistent states on the path are added to the *OPEN* list and their updated values are propagated through the state space. Because the current propagation phase may alter the solution path, the new solution path needs to be checked for underconsistent states. The entire process repeats until a solution path that contains only consistent states is returned.

## Applicability: Replanning Algorithms

Delayed D* has been shown to be significantly more efficient than D* Lite in certain domains (Ferguson & Stentz 2005). Typically, it is most appropriate when there is a relatively large distance between the start state and the goal state, and changes are being observed in arbitrary locations in the graph (for example, map updates are received from a satellite). This is because it is able to ignore the edge cost increases that do not involve its current solution path, which in these situations can lead to a dramatic decrease in overall computation. When a robot is moving towards a goal in a completely unknown environment, Delayed D* will not provide much benefit over D* Lite, as in this scenario typically the costs of only few states outside of the current least-cost path have been computed and therefore most edge cost increases will be ignored by both algorithms. There are also scenarios in which Delayed D* will do more processing than D* Lite: imagine a case where the processing of underconsistent states changes the solution path several times, each time producing a new path containing underconsistent states. This results in a number of replanning phases, each potentially updating roughly the same area of the state space, and will be far less efficient than dealing with all the underconsistent states in a single replanning episode. However, in realistic navigation scenarios, such situations are very rare.

In practise, both D* Lite and Delayed D* are very effective for replanning in the context of mobile robot navigation. Typically, in such scenarios the changes to the graph are happening close to the robot (through its observations), which means their effects are usually limited. When this is the case, using an incremental replanner such as D* Lite will be far more efficient than planning from scratch. However, this is not universally true. If the areas of the graph being changed are not necessarily close to the position of the robot, it is possible for D* Lite to be *less* efficient than A*. This is because it is possible for D* Lite to process every state in the environment twice: once as an underconsistent state and once as an overconsistent state. A*, on the other hand, will only ever process each state once. The worst-case scenario for D* Lite, and one that illustrates this possibility, is when changes are being made to the graph in the vicinity of the goal. It is thus common for systems using D* Lite to abort the replanning process and plan from scratch whenever either major edge cost changes are detected or some predefined threshold of replanning processing is reached.

Also, when navigating through completely unknown environments, it can be much more efficient to search forwards from the agent position to the goal, rather than backwards from the goal. This is because we typically assign optimistic costs to edges whose costs we don't know. As a result, areas of the graph that have been observed have more expensive edge costs than the unexplored areas. This means that, when searching forwards, as soon as the search exits the observed area it can rapidly progress through the unexplored area directly to the goal. However, when searching backwards, the search initially rapidly progresses to the observed area, then once it encounters the more costly edges in the observed area, it begins expanding large portions of the unexplored area trying to find a cheaper path. As a result, it can be significantly more efficient to use A* rather than backwards A* when replanning from scratch. Because the agent is moving, it is not possible to use a forwards-searching incremental replanner, which means that the computational advantage of using a replanning algorithm over planning from scratch is reduced.

As mentioned earlier, these algorithms can also be applied to symbolic planning problems (Koenig, Furcy, & Bauer 2002; Liu, Koenig, & Furcy 2002). However, in these cases it is important to consider whether there is an available predecessor function in the particular planning domain. If not, it is necessary to maintain for each state $s$ the set of all states $s'$ that have used $s$ as a successor state during the search, and treat this set as the set of predessors of $s$. This is also useful when such a predecessor function exists but contains a very large number of states; maintaining a list of just the states that have actually used $s$ as a successor can be far more efficient than generating all the possible predecessors.

In the symbolic planning community it is also common to use inconsistent heuristics since problems are often infeasible to solve optimally. The extensions to D* Lite presented in (Likhachev & Koenig 2005) enable D* Lite to handle inconsistent heuristics. These extensions also allow one to vary the tie-breaking criteria when selecting states from the *OPEN* list for processing. This might be important when a problem has many solutions of equal costs and the *OPEN* list contains a large number of states with the same priorities.

Apart from the static approaches (Dijkstra's, A*), all of the algorithms that we discuss in this paper attempt to reuse previous results to make subsequent planning tasks easier. However, if the planning problem has changed sufficiently since the previous result was generated, this result may be a burden rather than a useful starting point.

For instance, it is possible in symbolic domains that altering the cost of a single operator may affect the path cost of

a huge number of states. As an example, modifying the cost of the *load* operator in the rocket domain may completely change the nature of the solution. This can also be a problem when path planning for robots with several degrees of freedom: even if a small change occurs in the environment, it can cause a huge number of changes in the complex configuration space. As a result, replanning in such scenarios can often be of little or no benefit.

## Anytime Algorithms

When an agent must react quickly and the planning problem is complex, computing optimal paths as described in the previous sections can be infeasible, due to the sheer number of states required to be processed in order to obtain such paths. In such situations, we must be satisfied with the best solution that can be generated in the time available.

A useful class of deterministic algorithms for addressing this problem are commonly referred to as *anytime* algorithms. Anytime algorithms typically construct an initial, possibly highly suboptimal, solution very quickly, then improve the quality of this solution while time permits (Zilberstein & Russell 1995; Dean & Boddy 1988; Zhou & Hansen 2002; Likhachev, Gordon, & Thrun 2003; Horvitz 1987). Heuristic-based anytime algorithms often make use of the fact that in many domains inflating the heuristic values used by A* (resulting in the weighted A* search) often provides substantial speed-ups at the cost of solution optimality (Bonet & Geffner 2001; Korf 1993; Zhou & Hansen 2002; Edelkamp 2001; Rabin 2000; Chakrabarti, Ghosh, & DeSarkar 1988). Further, if the heuristic used is consistent[2], then multiplying it by an inflation factor $\epsilon > 1$ will produce a solution guaranteed to cost no more than $\epsilon$ times the cost of an optimal solution. Likhachev, Gordon, and Thrun use this property to develop an anytime algorithm that performs a succession of weighted A* searches, each with a decreasing inflation factor, where each search reuses efforts from previous searches (Likhachev, Gordon, & Thrun 2003). Their approach provides suboptimality bounds for each successive search and has been shown to be much more efficient than competing approaches (Likhachev, Gordon, & Thrun 2003).

Likhachev et al.'s algorithm, Anytime Repairing A* (ARA*), limits the processing performed during each search by only considering those states whose costs at the previous search may not be valid given the new $\epsilon$ value. It begins by performing an A* search with an inflation factor $\epsilon_0$, but during this search it only expands each state at most once[3]. Once a state $s$ has been expanded during a particular search, if it becomes inconsistent (i.e., $g(s) \neq rhs(s)$) due to a cost change associated with a neighboring state, then it is not reinserted into the queue of states to be expanded. Instead, it is placed into the *INCONS* list, which contains all inconsistent states already expanded. Then, when the current search terminates, the states in the *INCONS* list are inserted into a fresh priority queue (with new priorities based on the new $\epsilon$ inflation factor) which is used by the next search. This improves the efficiency of each search in two ways. Firstly, by only expanding each state at most once a solution is reached much more quickly. Secondly, by only reconsidering states from the previous search that were inconsistent, much of the previous search effort can be reused. Thus, when the inflation factor is reduced between successive searches, a relatively minor amount of computation is required to generate a new solution.

A simplified, backwards-searching version of the algorithm is given in Figure 4[4]. Here, the priority of each state $s$ in the *OPEN* queue is computed as the sum of its cost $g(s)$ and its inflated heuristic value $\epsilon \cdot h(s_{start}, s)$. *CLOSED* contains all states already expanded once in the current search, and *INCONS* contains all states that have already been expanded and are inconsistent.

**key**$(s)$
01. return $g(s) + \epsilon \cdot h(s_{start}, s)$;

**ImprovePath**()
02. while ($\min_{s \in OPEN}$(key($s$)) < key($s_{start}$))
03. remove $s$ with the smallest key($s$) from *OPEN*;
04. *CLOSED* = *CLOSED* $\cup \{s\}$;
05. for all $s' \in Pred(s)$
06. if $s'$ was not visited before
07. $g(s') = \infty$;
08. if $g(s') > c(s', s) + g(s)$
09. $g(s') = c(s', s) + g(s)$;
10. if $s' \notin$ *CLOSED*
11. insert $s'$ into *OPEN* with key($s'$);
12. else
13. insert $s'$ into *INCONS*;

**Main**()
14. $g(s_{start}) = \infty$; $g(s_{goal}) = 0$;
15. $\epsilon = \epsilon_0$;
16. *OPEN* = *CLOSED* = *INCONS* = $\emptyset$;
17. insert $s_{goal}$ into *OPEN* with key($s_{goal}$);
18. ImprovePath();
19. publish current $\epsilon$-suboptimal solution;
20. while $\epsilon > 1$
21. decrease $\epsilon$;
22. Move states from *INCONS* into *OPEN*;
23. Update the priorities for all $s \in$ *OPEN* according to key($s$);
24. *CLOSED* = $\emptyset$;
25. ImprovePath();
26. publish current $\epsilon$-suboptimal solution;

Figure 4: **The ARA* Algorithm (backwards version).**

## Applicability: Anytime Algorithms

ARA* has been shown to be much more efficient than competing approaches and has been applied successfully to path planning in high-dimensional state spaces, such as kinematic robot arms with 20 links (Likhachev, Gordon, & Thrun 2003). It has thus effectively extended the applicability of

[2] A (forwards) heuristic $h$ is consistent if, for all $s \in S$, $h(s, s_{goal}) \leq c(s, s') + h(s', s_{goal})$ for any successor $s'$ of $s$, and $h(s_{goal}, s_{goal}) = 0$.

[3] It is proved in (Likhachev, Gordon, & Thrun 2003) that this still guarantees an $\epsilon_0$ suboptimality bound.

[4] The backwards-searching version is shown because it will be useful when discussing the algorithm's similarity to D* Lite.


Figure 5: The ATRV robotic platform.

deterministic planning algorithms into much higher dimensions than previously possible. It has also been used to plan smooth trajectories for outdoor mobile robots in known environments. Figure 5 shows an outdoor robotic system that has used ARA* for this purpose. Here, the search space involved four dimensions: the $(x, y)$ position of the robot, the robot's orientation, and the robot's velocity. ARA* is able to plan suboptimal paths for the robot very quickly, then improve the quality of these paths as the robot begins its traverse (as the robot moves the start state changes and therefore in between search iterations the heuristics are recomputed for all states in the *OPEN* list right before their priorites are updated).

ARA* is well suited to domains in which the state space is very large and suboptimal solutions can be generated efficiently. Although using an inflation factor $\epsilon$ usually expedites the planning process, this is not guaranteed. In fact, it is possible to construct pathological examples where the best-first nature of searching with a large $\epsilon$ can result in much longer processing times. The larger $\epsilon$ is, the more greedy the search through the space is, leaving it more prone to getting temporarily stuck in local minima. In general, the key to obtaining anytime behavior with ARA* is finding a heuristic function with shallow local minima. For example, in the case of robot navigation a local minimum can be a U-shaped obstacle placed on the straight line connecting a robot to its goal (assuming the heuristic function is Euclidean distance) and the size of the obstacle determines how many states weighted A*, and consequently ARA*, will have to process before getting out of the minimum.

Depending on the domain one can also augment ARA* with a few optimizations. For example, in graphs with considerable branching factors the *OPEN* list can grow prohibitively large. In such cases, one can borrow an interesting idea from (Zhou & Hansen 2002) and prune (and never insert) the states from the *OPEN* list whose priorities based on un-inflated heuristic are already larger than the cost of the current solution (e.g., $g(s_{goal})$ in the forwards-searching version).

However, because ARA* is an anytime algorithm, it is only useful when an anytime solution is desired. If a solution with a particular suboptimality bound of $\epsilon_d$ is desired, and no intermediate solution matters, then it is far more efficient to perform a weighted A* search with an inflation factor of $\epsilon_d$ than to use ARA*.

Further, ARA* is only applicable in static planning domains. If changes are being made to the planning graph, ARA* is unable to reuse its previous search results and must replan from scratch. As a result, it is not appropriate for dynamic planning problems. It is this limitation that motivated research into the final set of algorithms we discuss here: anytime replanners.

## Anytime Replanning Algorithms

Although each is well developed on its own, there has been relatively little interaction between the above two areas of research. Replanning algorithms have concentrated on finding a single, usually optimal, solution, and anytime algorithms have concentrated on static environments. But some of the most interesting real world problems are those that are both dynamic (requiring replanning) *and* complex (requiring anytime approaches).

As a motivating example, consider motion planning for a kinematic arm in a populated office area. A planner for such a task would ideally be able to replan efficiently when new information is received indicating that the environment has changed. It would also need to generate suboptimal solutions, as optimality may not be possible given limited deliberation time.

Recently, Likhachev et al. developed Anytime Dynamic A* (AD*), an algorithm that combines the replanning capability of D* Lite with the anytime performance of ARA* (Likhachev *et al.* 2005). AD* performs a series of searches using decreasing inflation factors to generate a series of solutions with improved bounds, as with ARA*. When there are changes in the environment affecting the cost of edges in the graph, locally affected states are placed on the *OPEN* queue to propagate these changes through the rest of the graph, as with D* Lite. States on the queue are then processed until the solution is guaranteed to be $\epsilon$-suboptimal.

The algorithm is presented in Figures 6 and 7[5]. AD* begins by setting the inflation factor $\epsilon$ to a sufficiently high value $\epsilon_0$, so that an initial, suboptimal plan can be generated quickly. Then, unless changes in edge costs are detected, $\epsilon$ is gradually decreased and the solution is improved until it is guaranteed to be optimal, that is, $\epsilon = 1$. This phase is exactly the same as for ARA*: each time $\epsilon$ is decreased, all inconsistent states are moved from *INCONS* to *OPEN* and *CLOSED* is made empty.

When changes in edge costs are detected, there is a chance that the current solution will no longer be $\epsilon$-suboptimal. If the changes are substantial, then it may be computationally expensive to repair the current solution to regain $\epsilon$-suboptimality. In such a case, the algorithm increases $\epsilon$ so

[5]As with D* Lite the optimizations presented in (Koenig & Likhachev 2002) can be used to substantially speed up AD* and are recommended for an efficient implementation of the algorithm.

```
key(s)
01. if (g(s) > rhs(s))
02.     return [min(g(s), rhs(s)) + ε · h(s_start, s); min(g(s), rhs(s)))];
03. else
04.     return [min(g(s), rhs(s)) + h(s_start, s); min(g(s), rhs(s)))];

UpdateState(s)
05. if s was not visited before
06.     g(s) = ∞;
07. if (s ≠ s_goal) rhs(s) = min_{s'∈Succ(s)}(c(s, s') + g(s'));
08. if (s ∈ OPEN) remove s from OPEN;
09. if (g(s) ≠ rhs(s))
10.     if s ∉ CLOSED
11.         insert s into OPEN with key(s);
12.     else
13.         insert s into INCONS;

ComputeorImprovePath()
14. while (min_{s∈OPEN}(key(s)) ≺ key(s_start) OR rhs(s_start) ≠ g(s_start))
15.     remove state s with the minimum key from OPEN;
16.     if (g(s) > rhs(s))
17.         g(s) = rhs(s);
18.         CLOSED = CLOSED ∪ {s};
19.         for all s' ∈ Pred(s) UpdateState(s');
20.     else
21.         g(s) = ∞;
22.         for all s' ∈ Pred(s) ∪ {s} UpdateState(s');
```

Figure 6: **Anytime Dynamic A*: ComputeorImprovePath function.**

```
Main()
01. g(s_start) = rhs(s_start) = ∞; g(s_goal) = ∞;
02. rhs(s_goal) = 0; ε = ε_0;
03. OPEN = CLOSED = INCONS = ∅;
04. insert s_goal into OPEN with key(s_goal);
05. ComputeorImprovePath();
06. publish current ε-suboptimal solution;
07. forever
08.     if changes in edge costs are detected
09.         for all directed edges (u, v) with changed edge costs
10.             Update the edge cost c(u, v);
11.             UpdateState(u);
12.     if significant edge cost changes were observed
13.         increase ε or replan from scratch;
14.     else if ε > 1
15.         decrease ε;
16.     Move states from INCONS into OPEN;
17.     Update the priorities for all s ∈ OPEN according to key(s);
18.     CLOSED = ∅;
19.     ComputeorImprovePath();
20.     publish current ε-suboptimal solution;
21.     if ε = 1
22.         wait for changes in edge costs;
```

Figure 7: **Anytime Dynamic A*: Main function.**

that a less optimal solution can be produced quickly. Because edge cost increases may cause some states to become underconsistent, a possibility not present in ARA*, states need to be inserted into the *OPEN* queue with a key value reflecting the minimum of their old cost and their new cost. Further, in order to guarantee that underconsistent states propagate their new costs to their affected neighbors, their key values must use admissible heuristic values. This means that different key values must be computed for underconsistent states than for overconsistent states.

By incorporating these considerations, AD* is able to handle both changes in edge costs and changes to the inflation factor $\epsilon$. Like the replanning and anytime algorithms we've looked at, it can also be slightly modified to handle the situation where the start state $s_{start}$ is changing, as is the case when the path is being traversed by an agent. This allows the agent to improve and update its solution path while it is being traversed.

## An Example[6]

Figure 8 presents an illustration of each of the approaches described in the previous sections employed on a simple grid world planning problem. In this example we have an eight-connected grid where black cells represent obstacles and white cells represent free space. The cell marked R denotes the position of an agent navigating this environment towards a goal cell, marked G (in the upper left corner of the grid world). The cost of moving from one cell to any non-obstacle neighboring cell is one. The heuristic used by each algorithm is the larger of the x (horizontal) and y (vertical) distances from the current cell to the cell occupied by the agent. The cells expanded by each algorithm for each subsequent agent position are shown in grey. The resulting paths are shown as grey arrows.

The first approach shown is (backwards) A*. The initial search performed by A* provides an optimal path for the agent. After the agent takes two steps along this path, it receives information indicating that one of the cells in the top wall is in fact free space. It then replans from scratch using A* to generate a new, optimal path to the goal. The combined total number of cells expanded at each of the first three agent positions is 31.

The second approach is A* with an inflation factor of $\epsilon = 2.5$. This approach produces an initial suboptimal solution very quickly. When the agent receives the new information regarding the top wall, this approach replans from scratch using its inflation factor and produces a new path (which happens to be optimal). The total number of cells expanded is only 19, but the solution is only guaranteed to be $\epsilon$-suboptimal at each stage.

The third approach is D* Lite, and the fourth is D* Lite with an inflation factor of $\epsilon = 2.5$. The bounds on the quality of the solutions returned by these respective approaches are equivalent to those returned by the first two. However, because D* Lite reuses previous search results, it is able to produce its solutions with far fewer overall cell expansions.

[6]This example and the ensuing discussion are borrowed from (Likhachev *et al.* 2005).

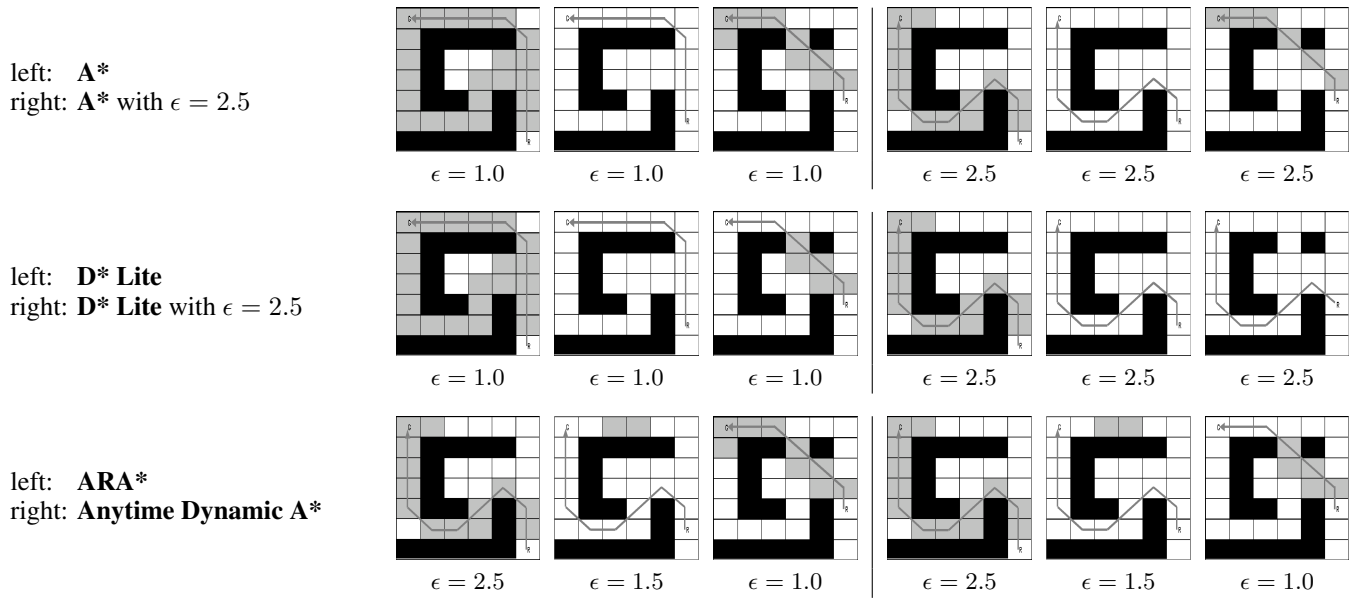



Figure 8: A simple robot navigation example. The robot starts in the bottom right cell and plans a path to the upper left cell. After it has moved two steps along its path, it observes a gap in the top wall. The states expanded by each of six algorithms (A*, A* with an inflation factor, D* Lite, D* Lite with an inflation factor, ARA*, and AD*) are shown at each of the first three robot positions. Example borrowed from (Likhachev *et al.* 2005).

D* Lite without an inflation factor expands 27 cells (almost all in its initial solution generation) and always maintains an optimal solution, and D* Lite with an inflation factor of 2.5 expands 13 cells but produces solutions that are suboptimal every time it replans.

The final row of the figure shows the results of (backwards) ARA* and AD*. Each of these approaches begins by computing a suboptimal solution using an inflation factor of $\epsilon = 2.5$. While the agent moves one step along this path, this solution is improved by reducing the value of $\epsilon$ to 1.5 and reusing the results of the previous search. The path cost of this improved result is guaranteed to be at most 1.5 times the cost of an optimal path. Up to this point, both ARA* and AD* have expanded the same 15 cells each. However, when the robot moves one more step and finds out the top wall is broken, each approach reacts differently. Because ARA* cannot incorporate edge cost changes, it must replan from scratch with this new information. Using an inflation factor of 1.0 it produces an optimal solution after expanding 9 cells (in fact this solution would have been produced regardless of the inflation factor used). AD*, on the other hand, is able to repair its previous solution given the new information and lower its inflation factor at the same time. Thus, the only cells that are expanded are the 5 whose cost is directly affected by the new information and that reside between the agent and the goal.

Overall, the total number of cells expanded by AD* is 20. This is 4 less than the 24 required by ARA* to produce an optimal solution, and substantially less than the 27 required by D* Lite. Because AD* reuses previous solutions in the same way as ARA* and repairs invalidated solutions in the same way as D* Lite, it is able to provide anytime solutions in dynamic environments very efficiently. The experimental evaluation on a simulated kinematic robot arm performed in (Likhachev *et al.* 2005) supports these claims and shows AD* to be many times more efficient than ARA*, to be able to operate under limited time constraints (an ability that D* Lite lacks), and finally to consistently produce significantly better solutions than D* Lite with inflated heuristics.

## Applicability: Anytime Replanning Algorithms

AD* has been shown to be useful for planning in dynamic, complex state spaces, such as 3 DOF robotic arms operating in dynamic environments (Likhachev *et al.* 2005). It has also been used for path-planning for outdoor mobile robots. In particular, those operating in dynamic or partially-known outdoor environments, where velocity considerations are important for generating smooth, timely trajectories. As discussed earlier, this can be framed as a path planning problem over a 4D state space, and an initial suboptimal solution can be generated using AD* in exactly the same manner as ARA*.

Once the robot starts moving along this path, it is likely that it will discover inaccuracies in its map of the environment. As a result, the robot needs to be able to quickly repair previous, suboptimal solutions when new information is gathered, then improve these solutions as much as possible given its processing constraints.

AD* has been used to provide this capability for two robotic platforms currently used for outdoor navigation: an ATRV and a Segway Robotic Mobility Platform (Segway RMP) (see Figure 9) (Likhachev *et al.* 2005). To direct the 4D search in each case, a fast 2D $(x, y)$ planner was used to provide the heuristic values.

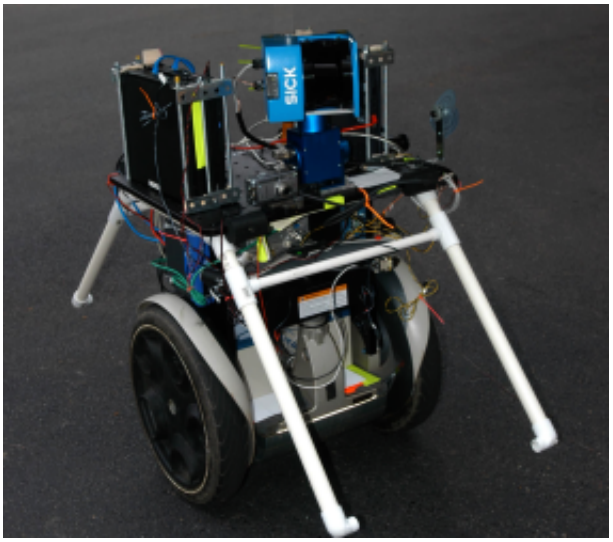



Figure 9: The Segway Robotic Mobility Platform.

Unfortunately, AD* suffers the drawbacks of both anytime algorithms *and* replanning algorithms. As with replanning algorithms, it is possible for AD* to be more computationally expensive than planning from scratch. In fact, this is even more so for AD*, since the version presented here and in (Likhachev *et al.* 2005) reorders the *OPEN* list every time $\epsilon$ is changed. It is thus important to have extra checks in place for AD* to prevent trying to repair the previous solution when it looks like it will be more time consuming than starting over (see lines 12 - 14 in Figure 7). For the outdoor navigation platforms mentioned above, this check is based on how much the 2D heuristic cost from the current state to the goal has changed based on changes to the map: if this change is large, there is a good chance replanning will be time consuming. In general it is worth taking into account how much of the search tree has become inconsistent, as well as how long it has been since we last replanned from scratch. If a large portion of the search tree has been affected and the last complete replanning episode was quite some time ago, it is probably worth scrapping the search tree and starting fresh. This is particularly true in very high-dimensional spaces where the dimensionality is derived from the complexity of the agent rather than the environment, since changes in the environment can affect a huge number of states.

There are also a couple optimizations that can be made to AD*. Firstly, it is possible to limit the expense of reordering the *OPEN* list each time $\epsilon$ changes by reducing the size of the queue. Specifically, *OPEN* can be split into a priority queue containing states with low key values and one or more unordered lists containing the states with very large key values. The states from the unordered lists need only be considered if the element at the top of the priority queue has a larger key value than the state with minimum key value in these lists. We thus need only maintain the minimum key value (or some lower bound for this value) for all states in the unordered lists. Another more sophisticated and potentially more effective idea that avoids the re-order operation altogether is based on adding a bias to newly inconsistent states (Stentz 1995) and is discussed in (Likhachev *et al.* 2005).

## Conclusions

In this paper, we have discussed a family of heuristic algorithms for path planning in real world scenarios. We have attempted to highlight the fundamental similarities between each of the algorithms, along with their individual strengths, weaknesses, and applicable problem domains. A common underlying theme throughout this discussion has been the variable value of previous solutions. When the problem being solved does not change significantly between invocations of our planner, it can be highly advantageous to take advantage of previous solutions as much as possible in constructing a new one. When the problem being solved does change, previous solutions are less useful, and can even be detrimental to the task of arriving at a new solution.

## Acknowledgments

The authors would like to thank Sven Koenig for fruitful discussions. This work was sponsored by DARPA's MARS program and the U.S. Army Research Laboratory, under contract "Robotics Collaborative Technology Alliance". The views contained in this document are those of the authors and do not represent the official policies or endorsements of the U.S. Government. Dave Ferguson is partially supported by a National Science Foundation Graduate Research Fellowship.

## References

Barbehenn, M., and Hutchinson, S. 1995. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest path trees. *IEEE Transactions on Robotics and Automation* 11(2):198–214.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

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

Chakrabarti, P.; Ghosh, S.; and DeSarkar, S. 1988. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence* 34:97–113.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Edelkamp, S. 2001. Planning with pattern databases. In *Proceedings of the European Conference on Planning*.

Ersson, T., and Hu, X. 2001. Path planning and navigation of mobile robots in unknown environments. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*.

Ferguson, D., and Stentz, A. 2005. The Delayed D* Algorithm for Efficient Path Replanning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Gelperin, D. 1977. On the optimality of A*. *Artificial Intelligence* 8(1):69–76.

Hart, P.; Nilsson, N.; and Rafael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE trans. Sys. Sci. and Cyb.* 4:100–107.

Hebert, M.; McLachlan, R.; and Chang, P. 1999. Experiments with driving modes for urban robots. In *Proceedings of SPIE Mobile Robots*.

Horvitz, E. 1987. Problem-solving design: Reasoning about computational value, trade-offs, and resources. In *Proceedings of the Second Annual NASA Research Forum*.

Huiming, Y.; Chia-Jung, C.; Tong, S.; and Qiang, B. 2001. Hybrid evolutionary motion planning using follow boundary repair for mobile robots. *Journal of Systems Architecture* 47:635–647.

Kavraki, L.; Svestka, P.; Latombe, J.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

Koenig, S., and Likhachev, M. 2002. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Koenig, S.; Furcy, D.; and Bauer, C. 2002. Heuristic search-based replanning. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling*, 294–301.

Korf, R. 1993. Linear-space best-first search. *Artificial Intelligence* 62:41–78.

LaValle, S., and Kuffner, J. 1999. Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

LaValle, S., and Kuffner, J. 2001. Rapidly-exploring Random Trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions* 293–308.

LaValle, S. 1998. Rapidly-exploring Random Trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa state University.

LaValle, S. 2005. *Planning Algorithms*. In progress - see http://msl.cs.uiuc.edu/planning/.

Likhachev, M., and Koenig, S. 2005. A Generalized Framework for Lifelong Planning A*. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Likhachev, M.; Ferguson, D.; Gordon, G.; Stentz, A.; and Thrun, S. 2005. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*. MIT Press.

Likhachev, M. 2003. Search techniques for planning in large dynamic deterministic and stochastic environments. Thesis proposal. School of Computer Science, Carnegie Mellon University.

Liu, Y.; Koenig, S.; and Furcy, D. 2002. Speeding up the calculation of heuristics for heuristic search-based planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 484–491.

Matthies, L.; Xiong, Y.; Hogg, R.; Zhu, D.; Rankin, A.; Kennedy, B.; Hebert, M.; Maclachlan, R.; Won, C.; Frost, T.; Sukhatme, G.; McHenry, M.; and Goldberg, S. 2000. A portable, autonomous, urban reconnaissance robot. In *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS)*.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Tioga Publishing Company.

Podsedkowski, L.; Nowakowski, J.; Idzikowski, M.; and Vizvary, I. 2001. A new solution for path planning in partially known or unknown environments for nonholonomic mobile robots. *Robotics and Autonomous Systems* 34:145–152.

Rabin, S. 2000. A* speed optimizations. In DeLoura, M., ed., *Game Programming Gems*, 272–287. Rockland, MA: Charles River Media.

Ramalingam, G., and Reps, T. 1996. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms* 21:267–305.

Stentz, A., and Hebert, M. 1995. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots* 2(2):127–145.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Stentz, A. 1995. The Focussed D* Algorithm for Real-Time Replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Thayer, S.; Digney, B.; Diaz, M.; Stentz, A.; Nabbe, B.; and Hebert, M. 2000. Distributed robotic mapping of extreme environments. In *Proceedings of SPIE Mobile Robots*.

Zhou, R., and Hansen, E. 2002. Multiple sequence alignment using A*. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. Student Abstract.

Zilberstein, S., and Russell, S. 1995. Approximate reasoning using anytime algorithms. In *Imprecise and Approximate Computation*. Kluwer Academic Publishers.

Zlot, R.; Stentz, A.; Dias, M.; and Thayer, S. 2002. Multi-robot exploration controlled by a market economy. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

# A Continuous Anytime Planning Module for an Autonomous Earth Watching Satellite

**Sylvain Damiani**
ONERA, Toulouse, France
Sylvain.Damiani@onera.fr

**Gérard Verfaillie**
LAAS-CNRS, Toulouse, France
Gerard.Verfaillie@laas.fr

**Marie-Claire Charmeau**
CNES, Toulouse, France
Marie-Claire.Charmeau@cnes.fr

**Abstract**

In this paper, we present the problem of management of an Earth watching mission (detection, observation, and tracking of forest fires and volcanic eruptions) by means of a constellation of low-orbit satellites. We show that the mission reactivity requirements and the communication constraints make on-board decision capabilities absolutely necessary. After showing how tracking tasks can be shared on the ground between satellites, we focus on on-board decision-making mechanisms. We show how a continuous anytime planning module can be designed to deal as optimally and reactively as possible with observation and data down-loading decisions. After an analysis of simulation results, we try to draw from this particular setting general lessons about continuous anytime decision-making mechanisms for permanent missions in dynamic unforeseeable environments.

## An Earth global watching mission

### Fire and eruption detection, observation, and tracking

The space mission we discuss in this paper has been provided to us by the French Space Agency (CNES, (Charmeau 2002)), in order to assess the interest and the feasibility of on-board autonomous planning and scheduling modules. Although it is not an actual mission yet, it is a realistic mission, inspired from the *Bird* (http://spacesensors.dlr.de/SE/bird/) and *Fuego* (Escorial, Tourne, & Reina 2001) projects.

The mission objectives are to *detect*, to *observe*, and to *track* forest *fires* or volcanic *eruptions*. More precisely, starting fires and eruptions must be automatically *detected*, *localized* and roughly *identified*. In case of detection of a fire or an eruption by a satellite, this satellite must immediately send an *alarm* to the concerned ground mission center and trigger an *observation* of the associated ground area. After that and as long as it is necessary, this area must be *tracked* by this satellite and by the other ones of the constellation i.e., *observed* as regularly as possible. After each observation, data must be *delivered* as early as possible to the concerned ground mission centers.



### A constellation of Earth watching satellites

To fulfill this mission, we assume to have at our disposal the following space and ground physical components:

1. a constellation of 12 identical *low-orbit* (LEO) *satellites*, arranged according to a *Walker* schema (Walker 1970): 3 orbital planes, each with an inclination angle of $47,5^\circ$ with regard to the polar axis, 4 satellites per orbital plan, evenly distributed on a circular orbit at an altitude of 700 km;
2. a set of 3 *geostationary* (GEO) *satellites* which together cover the whole Earth surface;
3. a set of ground *mission centers*, possibly dedicated to a specific area and to a specific kind of event (either forest fire, or volcanic eruption).
4. a ground constellation *control center*.

Given their altitude, the LEO satellites have a revolution period round the Earth of about 100 minutes. Figure 1 is a schematic 3D view of the movement of the constellation within a 25 minute period. It represents the trajectory of each LEO satellite within this period with, for one of them, the ground strip that is swept by its detection instrument (see below). It represents also as examples three ground stations (only two are visible on this figure) with the cuts of their reception/emission cones at an altitude of 700km: a LEO satellite can receive or emit data from or to a station only when it is inside the associated circle.

Figure 2 is the 2D ground counterpart of Figure 1. For each LEO satellite, its represents the track on the ground of its trajectory within a 25 minute period and the ground strip that is swept by its detection instrument. Note the three orbital planes and the shift between the track of a satellite and the track of the following one in the same orbital plane, due to Earth rotation on itself. Simulations show that the time between two successive flights over a given ground area by any of the satellites of the constellation depends on the area latitude, but is very irregular at each latitude: from some minutes to some hours.

The GEO satellites can be used to relay *alarms* from the LEO satellites to the ground. At any time, each LEO satellite is covered by one of the 3 GEO ones and can thus send an alarm to it. From the GEO satellite, this alarm can be sent to the ground reception station associated with it, and

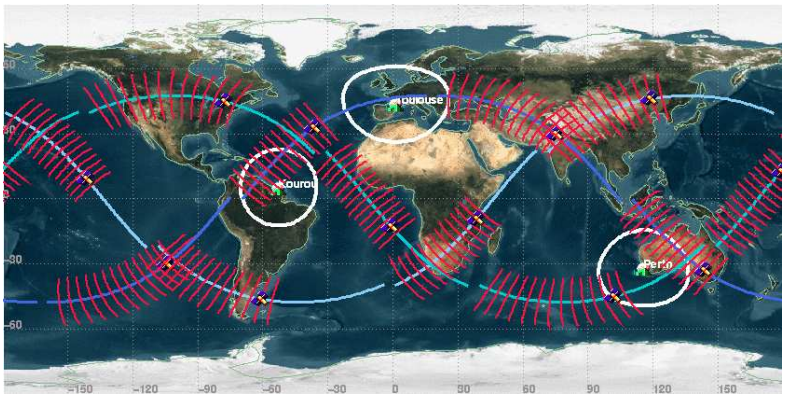



Figure 2: Track on the ground of the 12 satellites of the constellation within a 25 minute period.

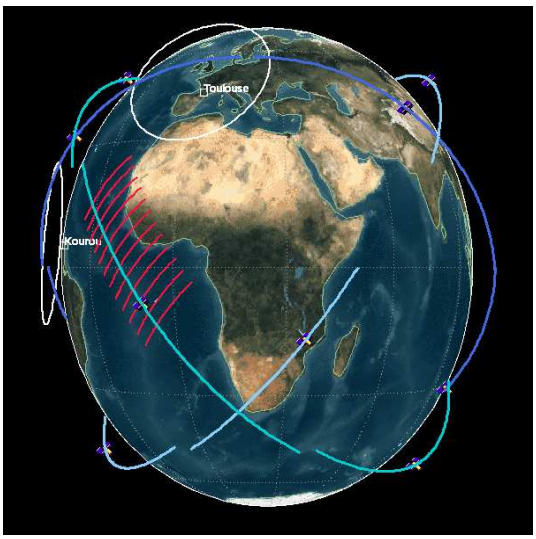



Figure 1: Movement of the constellation within a 25 minute period.

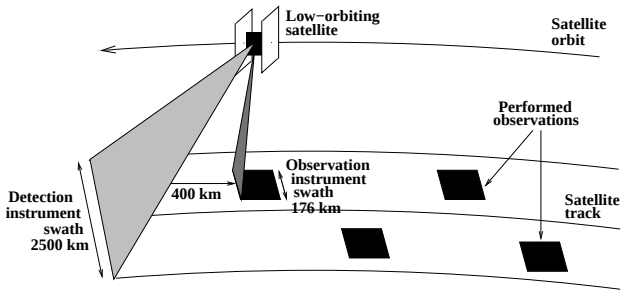



Figure 3: Detection and observation on-board each LEO satellite.

then to the concerned ground mission center via any ground communication network.

The ground mission centers can receive *observation data* from the LEO satellites, but only when they are in visibility of the associated reception station.

The ground constellation control center can send *observation requests* to the LEO satellites, but, as with mission centers, only when they are in visibility of the associated emission station.

## Detection and observation instruments

We assume that each LEO satellite is equipped with two instruments (see Figure 3):

1. an infrared *detection instrument*, the swath of which is 2500 km wide. This instrument is permanently active and pointed 30°, that is 400 km, in front of the satellite. Data analysis is instantly performed on board. In case of fire or eruption detection, an alarm is sent to the concerned ground mission center via the currently visible GEO satellite and an observation request is sent to the observation system;
2. an *observation instrument*, the swath of which is only 176 km wide. Four observation modes, in the visible, near infrared, and thermal infrared spectrums, are available, according to the kind of phenomenon to observe. This instrument is not permanently active. It is permanently pointed under the satellite, but a mobile mirror in front of it allows it to observe laterally any ground area in the strip that is swept by the detection instrument. Data that result from an observation are not analyzed on-board. They are down-loaded to the concerned ground mission center within visibility windows.

Note that, because the detection instrument is systematically pointed 30° in front of the satellite, there is an one minute delay between the detection of an unexpected phenomenon by a satellite and its possible observation by the same satellite.

Because the satellite can observe a ground area only when it arrives roughly at the same latitude, the starting and ending times of the observation of a given area from a given revolution of a given satellite are fixed and two areas the latitudes of which are too close may be *incompatible*: they cannot be observed by the same satellite from the same revolution be-

cause either a temporal overlapping, or an insufficient time to modify the mirror orientation (see Figure 4).

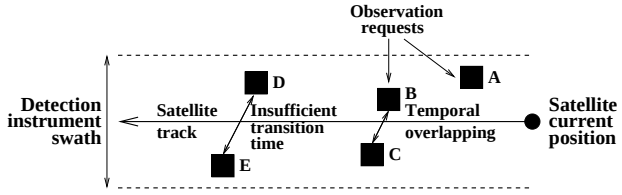



Figure 4: Incompatibilities between observations from the same satellite and the same revolution.

## On-board energy and memory

Each LEO satellite is limited in terms of *energy* and *memory* available on-board. Figure 5 shows the *permanent* and *temporary productions* and *consumptions* of energy and memory that must be taken into account.

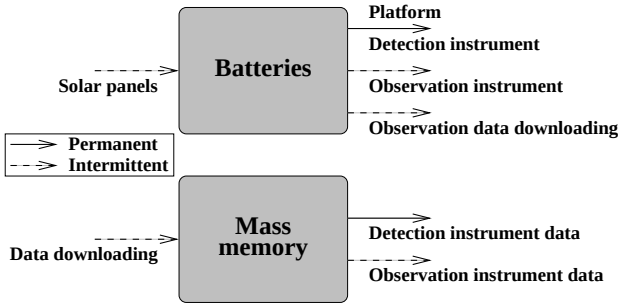



Figure 5: Productions and consumptions of energy and memory.

It is assumed that solar panels are powerful enough to cover the maximum energy consumption during day windows, but enough energy must be stocked into batteries to cover night windows. Energy and memory are not independent because observations consume energy and memory and because data down-loading produces memory (by releasing memory space), but consumes energy.

## Observation data down-loading and request up-loading

As previously said, *data down-loading* is possible from a LEO satellite to a ground mission center as soon as the satellite is in visibility of the reception station. But, at any time, a satellite cannot down-load data to more than one station and a station cannot receive data from more than one satellite.

Similarly, *request up-loading* is possible from the ground control center to a LEO satellite as soon as the satellite is in visibility of the emission station. But, at any time, the station cannot send requests to more than one satellite.

## Communication constraints

Figure 6 summarizes the communications that are possible between space and ground components.

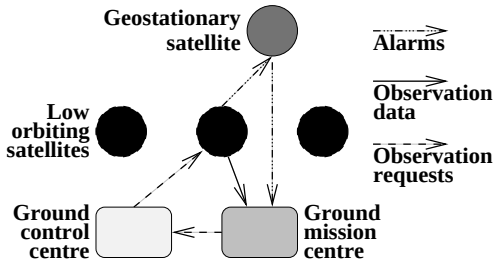



Figure 6: Possible communications between space and ground components.

It must be stressed that:

- communications between the LEO satellites and the ground, via the GEO satellites, are possible at any time, but limited to unidirectional low rate communications, only able to support alarms in case of fire or eruption detection;
- only direct visibility windows between LEO satellites and ground stations can be used for higher rate communications, able to support observation data down-loading and request up-loading.
- no direct communication is possible between LEO satellites;

Let us add that the time between two successive visibility windows between a given LEO satellite and a given ground station depends on the station latitude, but is very irregular along time: from 100 minutes (one revolution) to more than 15 hours.

# Decision-making organization between the ground constellation control center and the LEO satellites

Together, the global *mission objectives* and the *physical/technological setting* that have been presented so far strongly constrain the kind of *decision-making organization* that is possible between the ground constellation control center and the LEO satellites.

The first objective of the mission is to *detect* starting fires and eruptions and, in case of detection, to send alarms, to trigger observations, and to down-load observation data. Between a detection and the triggering of an observation of the associated area, there is only one minute. Between the observation and the associated data down-loading, there is a highly variable time that depends on the next visibility window between the satellite and the concerned ground mission center. This means that the satellite cannot wait for decisions from the ground control center (which could arrive only via visibility windows) to trigger an observation and to down-load associated data after detection. It must be able to make these decisions *autonomously* on-board and to manage for that possible conflicts with previously planned observations and data down-loadings.

The second objective of the mission is to *track* areas where fires or eruptions have been detected, by triggering

observations as regularly as possible and by down-loading associated data as early as possible. Because the time between two successive flights of a given satellite over a given ground area can go up to 15 hours, this task cannot be performed by a satellite alone, but by the whole constellation. Thus, tracking must be planned between all the constellation satellites. But, each satellite alone has only a partial view of the current fires and eruptions, and cannot communicate directly with the others. This turns out any choice for a decentralized task sharing mechanism between constellation satellites. In fact, via the alarms that are immediately relayed by the GEO satellites, the ground control center has at any time a complete view of all the fires or eruptions detected by all the constellation satellites. It can consequently *share* tracking tasks among satellites, with however two limitations. The first one is that it may be not aware of the actual state of each satellite, particularly of the actual levels of energy and memory available on-board. The second one is that it has no permanent communication with each satellite and that the result of its sharing will be sent to a given satellite only when this satellite will be in visibility. Both points do not rule out any interest in a sharing of the tracking tasks performed on the ground, but limit its impact: the result of the sharing shall be seen by LEO satellites only as *advice* or *requests*, not as orders; each LEO satellite shall remain able to deal *autonomously* with conflicts between requests coming either from the ground or from on-board detection, by taking into account its actual levels of energy and memory.

The organization that seems to fit the best the physical/technological setting at hand is thus a mix of *centralized* and *decentralized* decision-making: a *central entity* has at any time a global view of the work to do, shares this work between local entities, and communicates the sharing result to them when it can do it; each *local entity* does at any time the best it can, taking into account its state, the requests/advice of the central entity, and events that may occur unexpectedly.

This setting is strongly different from the one of *Earth observation*, which has been extensively studied for many years, in the setting of individual satellites and in the one of fleets or constellations of satellites (Sherwood *et al.* 1998; Bensana, Lemaître, & Verfaillie 1999; Wolfe & Sorensen 2000; Pemberton 2000; Vasquez & Hao 2001; Frank *et al.* 2001; Lemaître *et al.* 2002; Globus *et al.* 2002). In Earth observation, there is no detection on-board and all the requests come from the ground. This is why there has been no very strong interest in the design of autonomous decision-making capabilities, with only some exceptions (Bornschlegl, David, & Schindler 1998; Verfaillie & Bornschlegl 2000): observation plans can be built on the ground and regularly up-loaded to the satellites. But things change as soon as information is produced and analyzed on-board. This is the case when on-board decisions need information about the *actual state* of the satellite that is not accurately available on the ground at the planning time. This is also the case when on-board detection of the *actual cloud cover* allows the satellite to avoid useless observations in the visible spectrum (Lachiver *et al.* 2001), or when rough *on-board image analysis* allows the satellite to remove data associated with unusable observations, and thus to save on-board memory and to avoid useless data down-loading (Khatib *et al.* 2003; Morris *et al.* 2003). This is finally the case with *Earth watching*, because of the ability of the satellite to detect *new ground phenomena* and the need for *immediate reaction* in case of detection (Chien *et al.* 2004).

## Ground sharing of tracking tasks

Because this topic is not central in this paper, we offer only a global view of the way this sharing can be performed on the ground and we focus on its output: the requests that are sent to the LEO satellites.

Because requests can be up-loaded to a LEO satellite only when this satellite is in visibility of the ground control center, the control center must prepare requests to this satellite just before a visibility window and for the period between this window and the next one. For this period, it must share tracking tasks among all the satellites, taking into account the fact that, for the other satellites, it will be able to up-load requests to them only later, when they will be in visibility: for these satellites, requests cannot be modified till their next visibility window (see Figure 7).

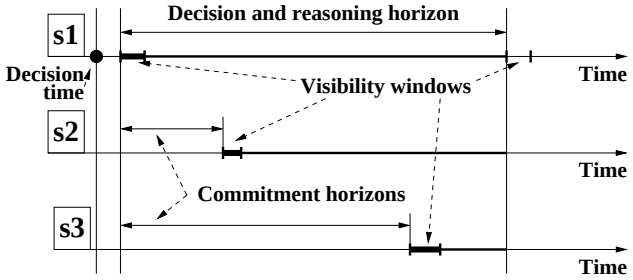



Figure 7: Decision time, commitment, decision and reasoning horizons for the sharing module just before a visibility window for the satellite $s_1$.

With each area where a fire or an eruption has been detected, is associated a *tracking request*. With each one, are associated a priority level $p,\ 1 \leq p \leq p_{max} - 1$, a tracking starting time $st$, and a tracking period $tp$. Ideally, this area should be observed at each time $st + i \cdot tp,\ i \geq 0$ and data should be down-loaded immediately after observation. In reality, even by using all the constellation satellites and all the ground stations, one can be only nearing this objective (see Figure 8).

The objective is then to assign each observation of each tracking request one (or more) satellite able to perform it, in such a way that all the tracking requests are satisfied as best as possible : observation times as close as possible to the reference times, data down-loading times as close as possible to the observation times. We associate an *observation note* and a *down-loading note* with each candidate local assignment. These notes are normalized in order to be compared. The note of a candidate local assignment is defined as the minimum of its observation note and of its down-loading note. We use then *lexicographic* and *leximin* orderings to compare two candidate global assignments, and *randomized greedy*

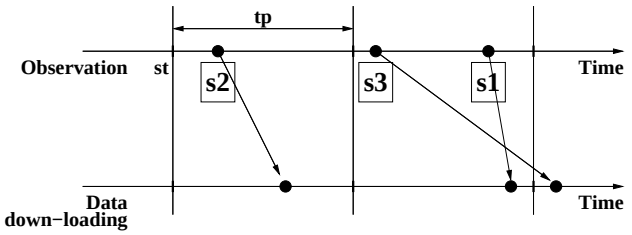



Figure 8: Tracking of a ground area: objective and example of assignment.

algorithms, inspired from (Bresina 1996), to produce good quality global assignments. Note that only direct incompatibilities between observations by the same satellite from the same revolution are taken into account when checking the consistency of a candidate global assignment.

The result is, for each satellite $s$, a set $R(s)$ of observation requests. With each one, is associated a *priority level* $p$, $1 \leq p \leq p_{max}-1$, inherited from the associated tracking request. In fact, when sending $R(s)$, the control center sends also the set $P$ of ground phenomena of which it is currently aware. If satellite $s$ detects a ground phenomenon which is in $P$ (already known) and the observation of which is not in $R(s)$ (not assigned to $s$), it assigns it the lowest priority level 0, which can be interpreted as *to do only if nothing other to do*. If it detects a ground phenomenon which is not in $P$ (not already known), it assigns it the highest priority level $p_{max}$.

Note that the sharing mechanism should take care of a reasonable *sharing* of the observations to perform between all the constellation satellites, in order to get no overloaded satellite, because an overloaded satellite might be compelled to ignore some ground requests in order to satisfy on-board high priority requests resulting from the detection of new ground phenomena. Inversely, if there are only few tracking requests, the control center could decide to assign each observation more than one satellite, in order to get more frequent observations and above all to be sure that at least one of the satellites performs it successfully.

## On-board decision-making organization

Because the detection instrument is permanently active and has only one working mode, the only decisions to make on-board are related to the use of the observation instrument (to trigger observations), of the mass memory (to record or to remove data), and of the antennas (to down-load data).

Although the management of observations and the one of resulting data physically interfere, we present them first separately, at least for the sake of clarity. Then, we show how they can be maintained separate, while guaranteeing consistency between decisions.

### Observation decisions

Let us recall that each LEO satellite is provided at any time with a set of observation requests, coming either from the ground via the visibility windows, or from the on-board detection at any time. With each request $r$, are associated a *priority* level $p(r)$, $0 \leq p(r) \leq p_{max}$, an *energy* consumption $e(r)$, and a *memory* consumption $m(r)$. The starting and ending times of the associated observation is completely determined by the geographical position of the target ground area.

The basic problem is then, just before the starting time of each candidate observation, to decide upon its triggering or not. This decision must be made by taking into account not only the priority of this observation and the ability to trigger it (eventual observation in progress, current mirror orientation, current energy and memory levels), but also the impact of this decision on future possible observations. This implies to reason as far as possible ahead. But, how to set the length of this ahead *reasoning horizon*? Roughly speaking, the larger it is, the more precisely assessed the impact of the current decision is, but the more uncertain data are, and the more time consuming the reasoning process is.

The choice we made is to design an *anytime* reasoning mechanism which adapts itself to the time it has at its disposal. Because candidate observations can be ordered according to their starting time, the reasoning horizon at step $i$ is the sequence made of the first $i$ candidate observations and the problem is to extract from this sequence a optimal consistent sub-sequence. When the reasoning process is started or restarted, it begins with a horizon of length 1: only the next candidate observation is taken into account. When reasoning at step $i$ is finished and time is still available for reasoning, the length of the reasoning horizon is incremented: the $(i+1)$th candidate observation is now taken into account.

The main advantage of such an approach is that a decision is available at any time, in fact as soon as the reasoning at step 1 is finished, that is very quickly. This decision is at any time the first candidate observation in the last computed optimal consistent sub-sequence: at step $i-1$ if reasoning is stopped when reasoning at step $i$. Although this is not always the case (Pearl 1983), we may expect that the quality of this decision increases with the length of the considered horizon, that is with the time available for reasoning.

This iterative mechanism is illustrated by Figure 9: after reasoning at step 7, the optimal sequence of observations is $\{1, 4, 6\}$ and the associated decision is 1, but after reasoning at step 8, the optimal sequence of observations is $\{2, 5, 8\}$ and the associated decision is now 2.

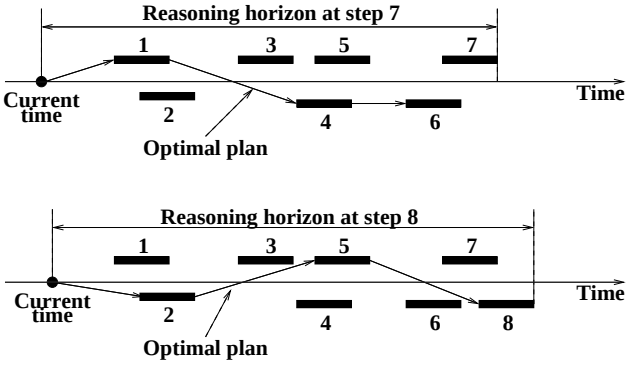



Figure 9: Reasoning on larger and larger horizons.

To reason, we need to compare two consistent subsequences. This is done by associating with any subsequence an *evaluation vector* that indicates for each priority level $p$ the number of selected observations of priority $p$ and by comparing two vectors lexicographically[1].

In fact, provided that energy and memory levels are discretized, the whole iterative mechanism can be implemented using a *dynamic programming* approach[2]. This approach is based on the recursive computing of the optimal evaluation vector $V^*(i, e, m)$ that can be associated with any candidate observation $i$, any possible level of energy $e$, and any possible level of memory $m$, and represents the best that can be obtained from the current time to the ending time of $i$, by going out of $i$ with an energy level $e$ and a memory level $m$. Equations 1, 2, 3, 4, and 5 are a simplified version of the equations that are used by this recursive computing. In these equations, $n$ is the current number of requests, $e_{min}$, $e_{max}$, $m_{min}$, and $m_{max}$ are the minimum and maximum levels of energy and memory, $e_0$ and $m_0$ are the current levels of energy and memory, $e(i)$ and $m(i)$ are the consumptions of $i$ in terms of energy and memory, $C(i)$ is the set of observations that start before $i$ and are compatible with it (no overlapping and sufficient transition time[3]), $\emptyset$ is a special vector used to represent what is impossible[4], 0 is a vector only made of 0s. $v(i)$ is a vector made of 0s but one 1 for the priority level of $i$, $V^*(i)$ is the best that can be obtained with the set of the observations that start before $i$ ($i$ included), $+$ represents the addition of two vectors, and $max$ represents the maximum using a lexicographic comparison.

Real equations take into account the actual productions and consumptions of energy and memory, as far as they can be forecast (permanent consumptions, day periods of energy production, energy consumption and memory production resulting from data-downloading).

$$\begin{aligned} &\forall i,\ 1 \leq i \leq n, \\ &\forall e,\ e_{min} \leq e \leq e_{max} - e(i), \\ &\forall m,\ m_{min} \leq m \leq m_{max} - m(i), \\ &V^*(i, e, m) = max(\emptyset, max_{j \in C(i)} \\ &[v(i) + V^*(j, e + e(i), m + m(i)]) \end{aligned} \tag{1}$$

$$\begin{aligned} &\forall i,\ 1 \leq i \leq n, \\ &\forall e,\ e_{min} \leq e \leq e_{max}, \\ &\forall m,\ m_{min} \leq m \leq m_{max}, \\ &(e > e_{max} - e(i)) \vee (m > m_{max} - m(i)) \\ &\Rightarrow V^*(i, e, m) = \emptyset \end{aligned} \tag{2}$$

$$V^*(0, e_0, m_0) = 0 \tag{3}$$

$$\begin{aligned} &\forall e,\ e_{min} \leq e \leq e_{max}, \\ &\forall m,\ m_{min} \leq m \leq m_{max}, \\ &(e \neq e_0) \vee (m \neq m_0) \\ &\Rightarrow V^*(0, e, m) = \emptyset \end{aligned} \tag{4}$$

$$V^*(i) = max_{j \leq i, e, m} V^*(j, e, m) \tag{5}$$

The most interesting feature of this recursive computing is that the optimal vectors $V^*(i)$ can be computed one after the other, by starting with $i = 1$ and by using at any step $i$ what has been previously computed. Each time reasoning at step $i$ is finished with the computing of $V^*(i)$, a decision can be extracted from the associated optimal consistent subsequence.

Another interesting feature is its low *complexity*: at any step $i$, a quadratic function of $i$ and a linear function of the number of discretization steps used for energy and memory.

The last but not the least feature is the *optimality* of the result, at least in the restrictive frame resulting from the limitation of the reasoning horizon at any step $i$ and from the discretization of energy and memory.

This choice of dealing with observation decisions one after the other differs from the classical choice in planning and scheduling, which consists in building a plan over a generally large horizon $H$, in executing it as far as it remains valid, and in building another one over the next horizon before the end of $H$. The justifications for such a choice are mainly three: (1) we are in a dynamic unforeseeable environment where anything may happen at any time, (2) there is time for reasoning during observations which last some tens of seconds and between observations which are often at wide intervals, (3) the iterative mechanism we implemented is very efficient (see experimental results) and can be started and restarted at any time.

## Data down-loading decisions

The setting of data down-loading is significantly different from the one of observation. Whereas an observation lasts some tens of seconds, the associated data down-loading lasts only some seconds. Whereas observations are sparsely distributed along time, data down-loading is concentrated on visibility windows, with no transition time between two successive data down-loadings in a visibility window, but up to 15 hours or more between two successive visibility windows. Such a setting justifies different choices in terms of decision-making organization.

The basic problem we consider is, just before the starting time of each visibility window $w$, to decide upon the data that will be down-loaded to the ground mission center $c$ that will be in visibility. Let $d_{max}$ be the duration of $w$. Let $O$ be the set of observations whose data is currently memorized on-board and is dedicated to $c$. With each $o \in O$,

[1] For example, with 4 priority levels from 0 to 3, selection $\{8, 4, 7, 5\}$ is preferred to selection $\{6, 10, 6, 5\}$, because both contain the same number of observations of priority 3 (5), but the first one contains more observations of priority 2 (7) than the second one does (6). In this case, observations of priority 1 or 0 do not matter.

[2] In fact, the problem to solve can be seen as the search for one of the longest paths in a multi-partite oriented graph, where a node is associated with each triple $< i, e, m >$ and a partition with each value of $i$.

[3] $0 \in C(i)$ iff the current state of the satellite (eventual observation in progress, current mirror orientation) allows $i$ to be triggered.

[4] It is smaller than all the other vectors and the result of adding it to any other vector is itself (minimum absorbing element).

are associated a *priority* level $p(o),\ 0 \leq p(o) \leq p_{max}$ and a down-loading *duration* $d(o)$. It may be also interesting to consider the *down-loading note* $dn_1(o)$ of $o$ if it would be down-loaded in $w$ and its down-loading note $dn_2(o)$ if it would be down-loaded in the next visibility window of $c$[5].

For the sake of clarity, let us consider the case of an isolated visibility window (no overlapping with another visibility window). In this case, the problem is to extract from $O$ a consistent optimal selection $O' \subseteq O$, that is a simple mono-dimensional *knapsack* problem[6].

To compare two consistent selections, we associate with any selection $O' \subseteq O$ an *evaluation vector* that contains for each priority level $p$ the vector of the notes of all the candidate down-loadings of priority $p$: $dn_1(o)$ if $o \in O'$ and $dn_2(o)$ otherwise. Two vectors are compared *lexicographically* between priority levels and using a *leximin* comparator inside each priority level[7].

As with the observation problem studied in the previous section, provided that time is discretized, this problem can be solved using a *dynamic programming* approach (see for example (Skiena 1998)). This approach uses any arbitrary ordering of the candidate down-loadings. It is based on the recursive computing of the optimal evaluation vector $V^*(i,d)$ that can be associated with any candidate down-loading $i$ and any duration $d$, and represents the best that can be obtained from the first $i$ down-loadings by using exactly duration $d$ (sum of the down-loading durations equal to $d$). This recursive computing exploits Equations 6, 7, 8, 9, and 10. In these equations, $n$ is the number of candidate down-loadings, $d_{max}$ is the duration of the visibility window, $d(i)$ is the down-loading duration of $i$, $\emptyset$ is the special vector used to represent what is impossible, $0$ is the special vector used to represent an empty selection (down-loading note $dn_2(o)$ assigned to each candidate down-loading $o$), $v(i)$ is a vector where the down-loading note $dn_2(o)$ is assigned to each candidate down-loading $o$ except $i$ which is assigned the down-loading note $dn_1(i)$, $V^*$ is the best that can be obtained with the set of candidate down-loadings, $+$ applied to two vectors results in a vector whose each component is the maximum of the associated components in both vectors[8], and $max$ represents the maximum using a lexicographic comparator between priority levels and a *leximin* comparator inside each level, as explained above.

$$\begin{array}{l} \forall i,\ 1 \leq i \leq n, \\ \forall d,\ d(i) \leq d \leq d_{max}, \end{array} \tag{6}$$

$$\begin{array}{l} V^*(i,d) = \\ max(\emptyset, V^*(i-1,d), v(i) + V^*(i-1, d-d(i))) \end{array}$$

$$\begin{array}{l} \forall i,\ 1 \leq i \leq n, \\ \forall d,\ 0 \leq d < d(i), \\ V^*(i,d) = max(\emptyset, V^*(i-1,d)) \end{array} \tag{7}$$

$$V^*(0,0) = 0 \tag{8}$$

$$\begin{array}{l} \forall d,\ 0 < d \leq d_{max}, \\ V^*(0,d) = \emptyset \end{array} \tag{9}$$

$$V^* = max_{d \leq d_{max}} V^*(n,d) \tag{10}$$

As with observations, the optimal vectors $V^*(n,d)$ could be computed one after the other from 0 up to $d_{max}$, and used to extract an optimal down-loading decision in the first part of any duration $d$ of the visibility window. But, we decided not to use this anytime version, mainly because of the very low complexity of the complete computing: a linear function of the number of candidate down-loadings and of the number of discretization steps used for time. Note that, as with observations, the result is *optimal*, at least in the restrictive frame resulting from time discretization.

However, there exists another algorithmic option, often used to solve approximately *knapsack* problems, that is a *greedy* algorithm using three criteria to order candidate down-loadings: firstly the priority level $p(o)$ in a decreasing order, secondly the down-loading note $dn_2(o)$ in an increasing order, and thirdly the downloading duration $d(o)$ in an increasing order. This *greedy* algorithm is still quicker than the *dynamic programming* one. It is not optimal, but offers guarantees about the distance to the optimum. It could be consequently used each time new data are memorized in order to assess what could be downloaded in the next visibility windows, and then which amount of energy will be consumed and which amount of memory will be released in each of them: forecasts that are useful to decide about observations (see below). As to the *dynamic programming* algorithm, it could be used just before the starting time of a visibility window, in order to compute an optimal down-loading plan in this window.

## Two interconnected decision problems

We presented observation and data down-loading as separate decision problems. This was for the sake of clarity, but also because we made actually the choice of separate decision modules. The justifications for such a choice are mainly two:

1. *decision dynamics* are very different: different *decision times* (before each possible observation for observation, before each visibility window for down-loading), different *decision horizons* (the next observation for observation, the next visibility window for down-loading), different *reasoning horizons* (variable for observation, the next visibility window for down-loading);
2. separately, both problems are simple and can be dealt very *efficiently* via dynamic programming or greedy algorithms.

[5] These notes are functions of the distance between observation and down-loading.

[6] In the case of overlapped visibility windows, the problem to solve is no more a simple *knapsack* problem, it becomes a *scheduling problem* because not all the data down-loading orderings are consistent with the visibility windows.

[7] For example, for a given priority level, the note vector $\{0.5, 0.6, 0.2\}$ is preferred to the note vector $\{0.2, 0.9, 0.4\}$ because the worst note is the same in both vectors $(0.2)$, but, after removing the worst in both vectors, the worst in the first vector has a better note $(0.5)$ than the worst in the second vector $(0.4)$.

[8] Because $\forall o \in O,\ dn_2(o) \leq dn_1(o)$.

But, in reality, both problems are not independent for mainly two reasons:

1. the global objective of the mission is to track important ground areas as regularly as possible and to deliver data as soon as possible after observation: there is *one objective*, not two;
2. observation and data down-loading *interfere* through *energy* and *memory*: observation consumes energy and memory, and data down-loading consumes energy and produces memory.

Between both decision modules, consistency in terms of mission objective is obtained via the maintenance for each request of the same level of *priority* from ground tracking sharing to observation and data down-loading.

About energy and memory, it can be observed that the situation is not symmetric: observation consumes energy and memory, but data down-loading consumes energy and produces memory. If data down-loading is insufficient, on-board memory gets quickly full, either observation is no more possible, or previously recorded data is removed. In the opposite direction, if observation is insufficient, data down-loading remains possible. Data down-loading is thus a *bottleneck* for the whole system. This is why we decided to give it *priority* for the access to energy.

In such conditions, data down-loading can be decided independently from observation, taking into account only currently recorded data. As to observation, it must forecast the amount of energy that will be consumed and also the amount of memory that will be released by data down-loading during the next visibility windows, and guarantee that data down-loading will have always enough energy to do its job. This can be done at any time via the greedy algorithm presented above.

## Experiments

Concerning observation decisions, first experimental results have been presented in (Damiani, Verfaillie, & Charmeau 2004). But, we carried out since then more ambitious experiments, involving the whole constellation (12 satellites), a control center, and two mission centers (one dedicated to forest fires and the other one to volcanic eruptions), over a temporal horizon of 16 hours.

We assume (1) about one hundred ground areas to track that are known at the beginning of the simulation horizon and (2) about ten that appear during the simulation horizon (in both cases, fifty-fifty shared between forest fires and volcanic eruptions). We assume that the tracking of the first ones is shared by the control center between the satellites of the constellation and that the resulting observation requests of priority 0, 1, or 2 are sent to each satellite at the beginning of the simulation horizon. We assume also that the second ones are detected by any satellite when flying over them, resulting in observation requests of priority 3.

We compared three ways of managing observation requests, in fact three ways of deciding to trigger or not an observation just before its starting time:

- to apply a very simple *decision rule*: trigger it when it is physically possible ($DR_1$);
- to apply a bit less blind *decision rule*: trigger it when it is physically possible and not in conflict with a future observation of higher priority ($DR_2$);
- to use the result of the *anytime observation planning* module: trigger it when it is the first observation of the current plan ($AP$).

Moreover, in order to measure the distance to optimality, we compare these three *realistic* management options, with two *unrealistic* ones. The first one, we refer to as $SP$ for *super-powerful*, assumes that the observation planning module has each time enough time to reason from the current time to the end of the simulation horizon. The second one, we refer to as $SPO$ for *super-powerful* and *omniscient*, assumes in addition that it knows from the beginning of the simulation horizon what will happen over the whole simulation horizon i.e., the ground phenomena that will appear and the resulting observation requests.

For each of these management options (either realistic or not), for each satellite, and for each priority level, we measured the ratio between the number of performed observations and the number of observation requests.

Table 10 shows typical results obtained on one satellite for which energy and memory constraints strongly limit the number of requests that can be satisfied. Note immediately that the total number of satisfied requests (last column) does not change dramatically from $DR_1$ to $AP$ (from 82 to 88). What significantly changes is the distribution of these satisfied requests between priority levels. Decision rules $DR_1$ and $DR_2$, which do not take into account energy and memory, satisfy too many low priority observation requests that consume energy and memory and then prevent the satellite from satisfying later high priority observation requests [9]. As expected, $DR_2$, which is a bit less blind than $DR_1$, performs better than this latter does. As expected too, the anytime planning module $AP$, which takes into account energy and memory, performs better than $DR_2$ does. Surprisingly, despite of a reasoning horizon that is limited by real-time constraints, it performs almost as well as unrealistic super-powerful and omniscient planning modules ($SP$ and $SPO$): it only fails to satisfy one request of priority 3 and one of priority 2.

Figure 11 shows the evolution of the energy and of the memory available on-board resulting from the $AP$ management option. Sharp increases in energy, followed by a plateau, occur when the satellite goes from a night to a day period and sharp increases in memory occur when the satellite is in visibility of a reception station and can down-load data.

## Discussion

The presentation we made may give the impression of local decision problems, separately studied and solved via specialized algorithms, without any global view of the whole

[9] These poor results could be improved, at least with regard to memory constraints, by allowing the on-board memory management system to overwrite low priority data with high priority one, as suggested in (Khatib *et al.* 2003).

| Priority levels | 3 | 2 | 1 | 0 | all |
|---|---|---|---|---|---|
| $DR_1$ | 4/10 | 13/24 | 14/23 | 51/96 | 82/153 |
| $DR_2$ | 6/10 | 15/24 | 15/23 | 47/96 | 83/153 |
| $AP$ | 9/10 | 17/24 | 20/23 | 40/96 | 86/153 |
| $SP$ | 10/10 | 18/24 | 20/23 | 39/96 | 87/153 |
| $SPO$ | 10/10 | 18/24 | 20/23 | 40/96 | 88/153 |

Figure 10: Results obtained on one satellite: for each priority level and for all of them, ratio between the number of performed observations and the number of observation requests.

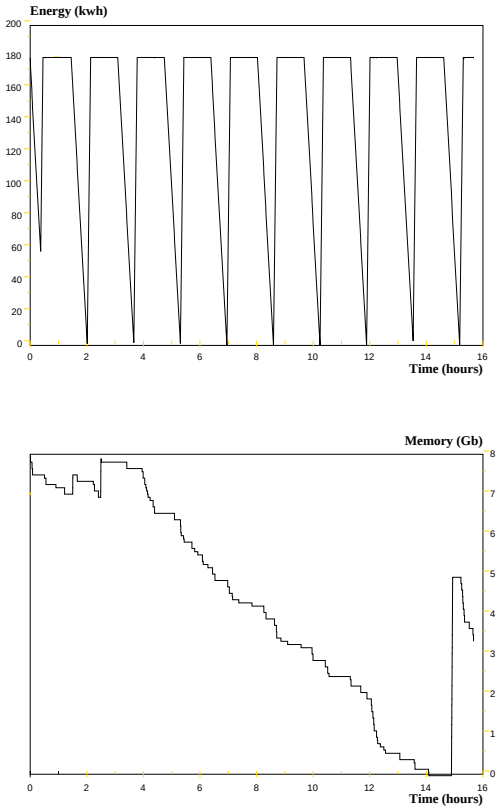



Figure 11: Evolution of the energy (top) and of the memory (bottom) available on-board.

decision system. As a matter of fact, such an impression is false, because each local decision problem has been studied according to the same following basic principles:

1. rather than a *proactive* approach such as for example the one of the *Markov Decision Processes* (Puterman 1994), a *reactive* approach is chosen to deal with uncertainty because of a *dynamic unforeseeable* environment: new ground phenomena, such as starting forest fires or volcanic eruptions, may occur at any time, but no usable model of these phenomena (where and when they may start) is available; if, for example, a probabilistic model would be available, the probabilities of a starting phenomenon would be always too small to have an influence on the decision; in such conditions, a sensible attitude consists in acting as if no new phenomenon should start and in being ready to react as best as possible each time a starting is detected;
2. each decision $d$ is made *as late as possible*, in order to be able to take into account the most up-to-date information;
3. for each decision $d$, the *decision horizon*, that is the temporal horizon on which actions will be decided (see Figure 12), is *as short as possible* in order not to commit to too large horizons and to be able to take into account new information and to make new decisions quickly after decision $d$ has been made and applied;
4. for each decision $d$, the *reasoning horizon*, that is the temporal horizon on which reasoning is performed to make decision $d$ (see Figure 12), is *as large as possible* in order to anticipate as well as possible system evolutions and to make as justified as possible decisions; note that uncertainty about the possible system evolutions may limit the interest in too large reasoning horizons;

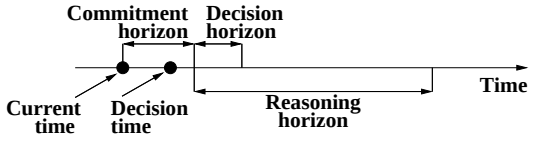



Figure 12: Commitment, decision, and reasoning horizons.

Note immediately a contradiction between the second and the fourth principle: if decisions are made as late as possible, the reasoning horizon may be limited because of a limited reasoning time. The solution we adopted is to use all the available time before decision time to reason and to prepare the decision: if nothing changes before decision time, decision is ready at decision time; if something changes, reasoning starts again with the new information. Such an approach, which can be seen as a form of *continuous* and *anytime* reasoning (Pollack & Horty 1999; Zilberstein 1996), is not limited to the application we dealt with. It can be *a priori* used for the control of any engine or system in a dynamic unforeseeable environment.

Let us end with the remark that such environments induce us to focus studies more on the smooth integration of planning and decision-making modules into the whole control

system (including situation recognition, supervision, and execution control modules) than on the planning task itself. That leads us to consider planning and scheduling more from the point of view of the correct behavior of the whole system than from the classical point of view of an isolated planning problem and of its optimal efficient solving.

## Acknowledgments

Thanks to Michel Lemaître, Félix Ingrand, Solange Lemai, and Frédérick Garcia for stimulating discussions about this work, and more generally about the best way of controlling a system in a dynamic uncertain environment.

## References

[Bensana, Lemaître, & Verfaillie 1999] Bensana, E.; Lemaître, M.; and Verfaillie, G. 1999. Earth Observation Satellite Management. *Constraints* 4(3):293–299.

[Bornschlegl, David, & Schindler 1998] Bornschlegl, E.; David, P.; and Schindler, H. 1998. On-Board Software and Operation for PROBA Small Satellite Autonomy. In *Proc. of the International Conference on Data Systems in Aerospace (DASIA-98)*.

[Bresina 1996] Bresina, J. 1996. Heuristic-Biased Stochastic Sampling. In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96)*, 271–278.

[Charmeau 2002] Charmeau, M. 2002. Mission de Référence pour le DE Autonomie. Note technique DTS/AE/SEA/IL/02-112, CNES.

[Chien *et al.* 2004] Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Lee, R.; Mandl, D.; Frye, S.; Trout, B.; Hengemihle, J.; D'Agostino, J.; Shulman, S.; Ungar, S.; Brakke, T.; Boyer, D.; VanGaasbeck, J.; Greeley, R.; Doggett, T.; Baker, V.; Dohm, J.; and Ip, F. 2004. The EO-1 Autonomous Science Agent. In *Proc. of the 3rd Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-04)*.

[Damiani, Verfaillie, & Charmeau 2004] Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2004. An Anytime Planning Approach for the Management of an Earth Watching Satellite. In *Proc. of the 4th International Workshop on Planning and Scheduling for Space (IWPSS-04)*.

[Escorial, Tourne, & Reina 2001] Escorial, D.; Tourne, I.; and Reina, F. 2001. FUEGO: A Dedicated Constellation of Small Satellites to Detect and Monitor Forest Fires. In *Proc. of the 3rd IAA Symposium on Small Satellites for Earth Observation*.

[Frank *et al.* 2001] Frank, J.; Jónsson, A.; Morris, R.; and Smith, D. 2001. Planning and Scheduling for Fleets of Earth Observing Satellites. In *Proc. of the 6th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-01)*.

[Globus *et al.* 2002] Globus, A.; Crawford, J.; Lohn, J.; and Morris, R. 2002. Scheduling Earth Observing Fleets Using Evolutionary Algorithms: Problem Description and Approach. In *Proc. of the 3rd NASA International Workshop on Planning and Scheduling for Space*.

[Khatib *et al.* 2003] Khatib, L.; Frank, J.; Smith, D.; Morris, R.; and Dungan, J. 2003. Interleaved Observation Execution and Rescheduling on Earth Observing Systems. In *Proc. of the ICAPS-03 Workshop on "Plan Execution"*.

[Lachiver *et al.* 2001] Lachiver, J.; Laherrère, J.; Sebbag, I.; Bataille, N.; and Bret-Dibat, T. 2001. System Feasibility of Onboard Clouds Detection and Observations Scheduling. In *Proc. of the 6th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-01)*.

[Lemaître *et al.* 2002] Lemaître, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology* 6:367–381.

[Morris *et al.* 2003] Morris, R.; Dungan, J.; Frank, J.; Khatib, L.; and Smith, D. 2003. An Integrated Approach to Earth Science Observation Scheduling. In *Proc. of the 3rd NASA Earth Science Technology Conference (ESTC-03)*.

[Pearl 1983] Pearl, J. 1983. On the Nature of Pathology in Game Searching. *Artificial Intelligence* 20:427–453.

[Pemberton 2000] Pemberton, J. 2000. Towards Scheduling Over-constrained Remote-sensing Satellites. In *Proc. of the 2nd NASA International Workshop on Planning and Scheduling for Space*, 84–89.

[Pollack & Horty 1999] Pollack, M., and Horty, J. 1999. There's More to Life Than Making Plans: Plan Management in Dynamic Multiagent Environments. *AI Magazine* 20(4):71–83.

[Puterman 1994] Puterman, M. 1994. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

[Sherwood *et al.* 1998] Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and Fukunaga, A. 1998. Using ASPEN to Automate EO-1 Activity Planning. In *Proc. of the IEEE Aerospace Conference*.

[Skiena 1998] Skiena, S. 1998. *The Algorithm Design Manual*. Telos/Springer-Verlag.

[Vasquez & Hao 2001] Vasquez, M., and Hao, J. 2001. A Logic-constrained Knapsack Formulation and a Tabu Algorithm for the Daily Photograph Scheduling of an Earth Observation Satellite. *Journal of Computational Optimization and Applications* 20(2):137–157.

[Verfaillie & Bornschlegl 2000] Verfaillie, G., and Bornschlegl, E. 2000. Designing and Evaluating an On-line Onboard Autonomous Earth Observation Satellite Scheduling System. In *Proc. of the 2nd NASA International Workshop on Planning and Scheduling for Space*, 122–127.

[Walker 1970] Walker, J. 1970. Circular Orbit Patterns Providing Whole Earth Coverage. Tech. Report 70211, Royal Aircraft Establishment.

[Wolfe & Sorensen 2000] Wolfe, W., and Sorensen, S. 2000. Three Scheduling Algorithms applied to the Earth Observing Systems Domain. *Management Science* 46(1):148–168.

[Zilberstein 1996] Zilberstein, S. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17(3):73–83.

# Enhancing the Anytime Behaviour of Mixed CSP-Based Planning

**Christophe Guettier**
SAGEM, 178, Rue de Paris, 91300 Massy, France
christophe.guettier@sagem.com

**Neil Yorke-Smith**
SRI International, Menlo Park, CA 94025, USA
nysmith@ai.sri.com

**Abstract**

An algorithm with the anytime property has an approximate solution always available; and the longer the algorithm runs, the better the solution becomes. Anytime planning is important in domains such as aerospace, where time for reasoning is limited and a viable (if suboptimal) course of action must be always available. In this paper we study anytime solving of a planning problem under uncertainty that arises from aerospace sub-system control. We examine an existing constraint model-based approach of the problem as a mixed constraint satisfaction problem (mixed CSP), an extension of classical CSP that accounts for uncontrollable parameters. We propose two enhancements to the existing decomposition algorithm: heuristics for selecting the next uncertain environment to decompose, and solving for incrementally longer planning horizons. We evaluate these enhancements empirically, showing that a heuristic on uncertainty analogous to 'first fail' gives the best performance, improving the anytime behaviour with respect to robustness to uncertainty. Further, we show that incremental horizon planning provides effective anytime behaviour with respect to plan executability, and that it can be combined with the decomposition heuristics.

## Introduction

The increasing desire for autonomy in aerospace systems, such as Uninhabited Aircraft Vehicles (UAVs), leads to increasing complexity in planning, scheduling, and control problems (Verfaillie 2001). Constraint-based techniques have proved effective for addressing such problems in the aerospace domain (e.g. (Muscettola *et al.* 1998; Allo *et al.* 2002; Frank & Jónsson 2004)). The real-world requirements of such problems mean that preferences, uncertainty, and dynamic change must be handled. For this, the classical constraint satisfaction problem (CSP) is inadequate. One extension to handle uncertainty is the mixed CSP framework, introduced by Fargier *et al.* (1995; 1996).

Our motivation comes from a problem in online planning of the control of an aerospace component such as a thruster. In order to enhance autonomous behaviour, the plan produced must take account of environmental uncertainty the aerospace system may encounter. During execution, the plan needs only specify the immediate next control action: thus continuous, incremental planning is possible for the problem. A constraint-based formulation as a mixed CSP was given in (Yorke-Smith & Guettier 2003); in it uncontrollable *parameters* model the uncertain evolution of physical quantities such as temperature.

An algorithm with the *anytime* property has an approximate solution always available; and the longer it runs, the better the solution becomes (Boddy & Dean 1994). If the algorithm is allowed to run to completion, a final solution is obtained. In the aerospace domain, planning is performed in an operational context where multiple levels of reactivity are required. This paper introduces an anytime solving approach to cope with uncertainty, in order to give more flexibility to the component management system during mission operations. Planning is invoked in the background, during cruise flight or in between critical phases of operation, and can be preempted at any time by higher priority tasks or by changes occurring at execution control levels. Thus the technique we described sits between and complements off-line planning and reactive component management.

This paper presents an experimental study of anytime planning with mixed CSPs. Specifically, we investigate the performance of the existing decomposition algorithm of (Fargier, Lang, & Schiex 1996) on our aerospace control planning problem as a case study. We describe two enhancements to the use of the algorithm designed to improve its anytime performance, and empirically assess their value. The two enhancements are decomposition heuristics for exploring the parameter space of uncertain environments, and incremental solving of the planning problem for successive horizons. The results show that a heuristic on uncertainty analogous to 'first fail' from the CSP literature gives the best performance, improving the anytime planning with respect to robustness to uncertainty. They also show that incremental horizon planning provides effective anytime behaviour with respect to plan executability, and that it can be combined with the decomposition heuristics.

## Background and Problem Domain

Our motivation for studying planning with mixed CSPs comes from a planning problem arising in the aerospace domain, formalized as the *Sub-system Control Planning Problem* (SCPP); a detailed description is found in (Yorke-Smith & Guettier 2003; Yorke-Smith 2004). The planning is situated in an online context where both mission status and situation awareness may evolve in multiple, non-controllable ways. Since contingent events may unexpectedly occur, a safe course of action for the system is required to be immedi-

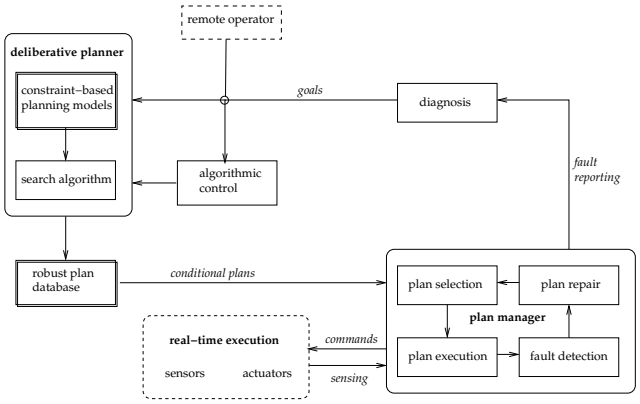



Figure 1: Outline architecture of a constraint-based agent.

ately available. The SCPP addresses deliberative plan generation in a dynamic context: deliberation is situated on board an autonomous system amid concurrent execution. Purely proactive, off-line planning is too slow, while purely reactive control provides a solution of too low quality.

Figure 1 illustrates how a constraint-based planning function can be integrated in an autonomous system, as the DS1 Experiment demonstrated (Muscettola *et al.* 1998).

Briefly, the input in an instance of the SCPP is a high-level description of an aerospace component, together with a description of the environmental uncertainty. A stated objective function on the performance of the component may also be given. The output sought is a plan for the commanding of the component. The plan must respect the behaviour guarantees that form part of the specification of the component. Further, as far as possible the plan must also optimize the performance of the component. The commanding is specified as a low-level automaton which models the component behaviour; for background, see (Arkin 1998).

For each aerospace component, an instance of the SCPP is parameterized by the planning horizon, $H \in \mathbb{N}$. Although the system plans supposing execution will begin at a future point in time, a decision can be demanded at any instant. Accordingly, during execution the plan needs only to specify the immediate next control action at the current horizon. During execution, the uncertain environment is observed just prior to each execution step.

Considered as a planning problem, the SCPP has: (1) nondeterministic actions due to contingent events, (2) fully observable states, and (3) semi-controllable effects, due to the environmental uncertainty. Planning consists of defining a consistent sequences of states in order to reach a given target state. This corresponds to the equipment changing modes of operation, in a feasible way, to reach a target mode. The timed sequence, the plan, must satisfy the model-based constraints and, possibly, optimize the performance function. As Figure 1 shows, the target mode is specified by mission and operational goals. For example, for a thruster component, the goal may be to achieve a certain thrust performance in a given time window, while maintaining the internal temperature within given limits.

The SCPP is stated as a unique rather than a sequential decision problem. Our solution is a conditional plan that covers the anticipated environmental uncertainty. As will be explained, this contingent plan corresponds to the conditional decision of a full observability mixed CSP. An execution step consists of selecting a plan branch according to the observed environment at that step. The proactive approach ensures a valid plan is available, however the environment evolves within anticipated limits, once deliberation is complete. Since planning and execution are concurrent, deliberation may not have time to complete before the next decision is demanded; hence the requirement for anytime planning.

## Mixed CSP and the Decomposition Algorithm

We now recall the mixed CSP framework and describe a constraint model-based representation of the SCPP as a mixed CSP. We then recall the algorithm presented in (Fargier, Lang, & Schiex 1996) for finding an optimal conditional decision in the case of full observability.

### Mixed CSP

Fargier *et al.* (1995; 1996) introduced the *mixed CSP* framework, an extension to the classical CSP for decision making with incomplete knowledge.[1] In a mixed CSP, variables are of two types: decision and non-decision variables. The first type are controllable by the agent: their values may be chosen. The second type, known as *parameters*, are uncontrollable: their values are assigned by extrogeneous factors. These factors are often referred to as 'Nature', meaning the environment, another agent, and so on.

Formally, a mixed CSP extends a classical finite domain CSP $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V}$ is a finite set of variables, $\mathcal{D}$ is the corresponding set of domains, and $\mathcal{C}$ is a set of constraints:

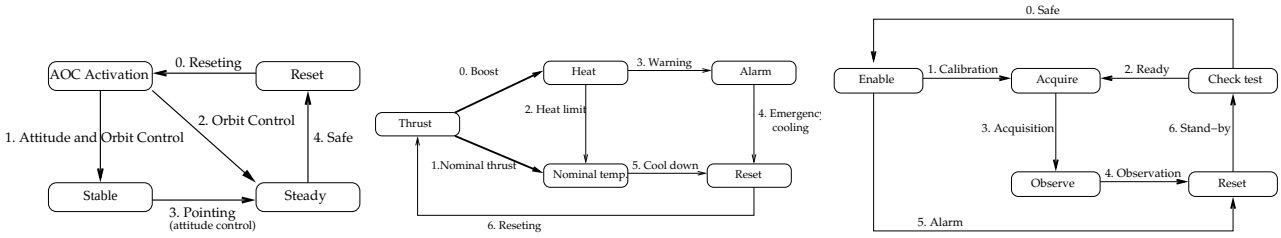
**Definition 1** *A* mixed CSP *is a 6-tuple* $\mathcal{P} = \langle \Lambda, L, V, D, \mathcal{K}, \mathcal{C} \rangle$ *where:*

- $\Lambda = \{\lambda_1, \ldots, \lambda_p\}$ *is a set of parameters*
- $L = L_1 \times \cdots \times L_p$*, where* $L_i$ *is the domain of* $\lambda_i$
- $V = \{x_1, \ldots, x_n\}$ *is a set of decision variables*
- $D = D_1 \times \cdots \times D_n$*, where* $D_i$ *is the domain of* $x_i$
- $\mathcal{K}$ *is a set of* data constraints *involving only parameters*
- $\mathcal{C}$ *is a set of constraints, each involving at least one decision variable and zero or more parameters*

A complete assignment of the parameters is a *realisation* (or world), and a complete assignment of the decision variables is a *decision* (or potential solution). A realisation is *possible* if the classical CSP $\langle \Lambda, L, \mathcal{K} \rangle$ is consistent, i.e. has at least one solution (otherwise the realisation is *impossible*). For every realisation $r$, the classical CSP $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ formed as the projection of $\mathcal{P}$ under realisation $\Lambda \leftarrow r$ is the *realised* (or candidate) problem induced by $r$ from $\mathcal{P}$. A realisation is *good* if the corresponding realised CSP is consistent (otherwise *bad*). We say a decision $d$ *covers* a realisation $r$ if $d$ is a solution to the realised CSP induced by $r$.

The nature of the outcome sought from a mixed CSP model depends on when knowledge about the state of the

[1] The earlier work (Fargier *et al.* 1995) associates a probability distribution with each parameter; we follow the later work in which a (discrete) uniform distribution is assumed.




(a) AOCS executive (b) Thruster (c) Star tracker

Figure 2: Discrete automata representing the behaviour of three spacecraft sub-systems

world will be acquired. If the realisation is observed before the decision must be made, we are in the case of *full observability*. In this case, the outcome sought is a *conditional decision* (or policy). This is a map between realisations and decisions that specifies, for a set of realisations $R$, a decision $d$ for each $r \in R$ such that $d$ covers $r$. We then say that the conditional decision *covers* $R$. Such a conditional decision is *optimal* if it covers every good realisation of $\mathcal{P}$. Deciding consistency of a binary mixed CSP is co-$\Sigma_2$ complete (Fargier, Lang, & Schiex 1996).

## Mixed CSP Model of a SCPP Instance

A constraint-based formulation of the SCPP is given in (Yorke-Smith & Guettier 2003). Following earlier work (Allo *et al.* 2002), this model-based formulation represents the component activity over a fixed discrete horizon, using a constraint-based non-deterministic finite state automaton. This automaton, synthesized automatically from the problem specification, is represented concretely as a mixed CSP. A conditional decision policy for the mixed CSP model corresponds to a viable plan for the SCPP instance.

Importantly, although the constraints may be complicated, we formulate the model such that each constraint involves at most one parameter. This reduces the complexity of building the plan, because computing which realisations are covered by a decision in the resulting mixed CSP is simplified (Fargier *et al.* 1995). Parameters arise from uncertain environment conditions, such as temperature variation, in each state of the automaton. The model includes constraints describing evolution of fluent physical quantities according to the environmental uncertainty, such as:

$$\Theta_{i+1} = E_j \times (\Theta_i + T_i \Delta_j) \quad (1)$$

where $\Theta_i$ and $T_i$ are discrete variables, $E_i$ are Boolean, and $\Delta_j$ are parameters. The details of the model are not central to this paper; they may be found in (Yorke-Smith 2004).

There may be an additional minimum performance requirement on feasible plans. This requirement corresponds to a percentage of the maximum possible performance $\text{perf}_{\text{max}}$ (which can be computed a priori); it is imposed as an additional hard constraint in the model:

$$\text{perf}(S) \geq k \times \text{perf}_{\text{max}} \quad (2)$$

where $\text{perf}(S)$ is the performance of a plan $S$, and $k \in [0, 1]$ is a given constant.

Figures 2(a)–2(c) show three discrete state automata. The automata represent the behaviour of three different, representative but simplified spacecraft sub-systems. They represent, respectively, an Attitude and Orbit Control System (AOCS), a thruster (Thruster), and a directional sensor (Tracker). While they bear some similarity in structure, the automata differ markedly in available actions and permitted timings, constraints, and evolution of fluent quantities such as energy (Yorke-Smith 2004). These differences impact strongly the problem difficulty, as exhibited by the experimental results that follow.

We build a mixed CSP model from the SCPP instance given by each automaton. Thus, the performance of solving these mixed CSPs will be the benchmark for our empirical study. These benchmarks are representative of some systems that are the source real-world of SCPP problems. Larger systems with more states and transitions can be envisaged, but they will not be necessarily formalized with automaton.

## Decomposition Algorithm

We say an algorithm has an *anytime property* (Boddy & Dean 1994) if: (1) an answer is available at any point in the execution of the algorithm (after some initial time, perhaps zero, required to provide a first valid solution); and (2) the quality of the answer improves with an increase in execution time. The *performance profile* of an algorithm is a curve that denotes the expected answer quality $Q(t)$ with execution time $t$ (Boddy & Dean 1989).

An algorithm to find an optimal conditional decision for a mixed CSP under full observability is presented in (Fargier *et al.* 1995; Fargier, Lang, & Schiex 1996). We call this the *decomposition algorithm* and denote it `decomp`. Because of the complexity of finding such a decision — both computational effort, and size of the outcome (in the worst case, one decision for every possible realisation) — `decomp` is designed as an anytime algorithm. Intuitively, it incrementally builds a list of decisions that eventually cover all good realisations. We omit discussion of some for us unnecessary subtleties about the algorithm.

Central to the method are sets of disjoint realisations called *environments*[2] and their judicious decomposition. Formally, an *environment* is a Cartesian product $l_1 \times \cdots \times l_p$,

[2]Environmental uncertainty should be distinguished from this technical definition of an *environment* as a set of realisations.

**Algorithm 1** Decomposition for an optimal cond. decision

1: $B \leftarrow \emptyset$ {*bad realisations*}
2: $D \leftarrow \emptyset$ {*decision–environment pairs*}
3: $E \leftarrow L_1 \times \cdots \times L_p$ {*environments still to be covered*}
4: **repeat**
5: Choose an environment $e$ from $E$ {*pick uncovered env.*}
6: **let** $\mathcal{C}_e$ **be** constraints that enforce $e$
7: **let** $P$ **be** the CSP $\langle \Lambda \cup \mathcal{V}, L \cup D, \mathcal{K} \cup \mathcal{C} \cup \mathcal{C}_e \rangle$
8: **if** $P$ is consistent **then**
9: **let** $s$ **be** a solution of $P$ {*find dec. covering* $\geq 1$ *real.*}
10: **let** $v$ **be** $s$ projected onto the domain variables $\mathcal{V}$
11: $R \leftarrow covers(v)$ {*find all realisations covered by* $v$}
12: Add the pair $(v, R)$ to $D$
13: $E \leftarrow \bigcup_{e' \in E} decompose(e', R)$ {*removed covered*}
14: **else** {*all realisations in* $e$ *impossible*}
15: Add $e$ to $B$
16: **until** $E = \emptyset$ {*all possible realisations covered*}
17: **return**$(B, D)$

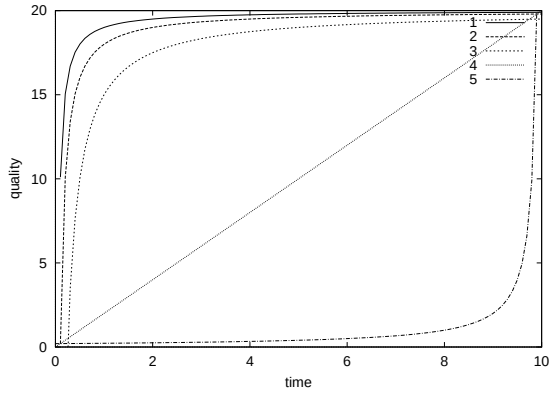



Figure 3: Idealized anytime performance profiles

where $l_i \subseteq L_i$: for example, if $L_1 = L_2 = \{a, b, c, d\}$, then an environment is $\{b, d\} \times \{c, d\}$. `decomp` is an anytime algorithm that incrementally computes successively closer approximations to an optimal decision. The number of realisations covered by the decision grows monotonically, and if allowed to finish without interruption, the algorithm returns an optimal conditional decision.

Pseudocode for `decomp` is given as Algorithm 1. (Fargier, Lang, & Schiex 1996) prove Algorithm 1 to be sound and complete: it eventually terminates, and if allowed to terminate, it returns a conditional decision that covers all good realisations. Moreover, if stopped at any point, $D$ contains decisions for (zero or more) good realisations and $B$ contains only bad realisations.

## Enhancing the Anytime Behaviour of `decomp`

Summarizing, we have described a model of our motivating problem as a mixed CSP, and recalled the algorithm we call `decomp` for a full observability mixed CSP. We now introduce two orthogonal extensions of `decomp` designed to improve its anytime performance for the requirements arising for planning and scheduling in the aerospace domain.

To see what we mean by improved anytime behaviour, consider the performance profiles shown in Figure 3. The horizontal axis depicts time $t$ and the vertical axis solution quality $Q(t)$. The straight line 4 represents the behaviour of an algorithm that monotonically increases solution quality at a constant rate. The curves 1–3 depict better anytime behaviour than 4, with 1 the best, because solution quality rises more sharply earlier in the solving. In contrast, curve 5 depicts a poor anytime behaviour. Thus moving from 4 to 2, for instance, is an improvement in anytime behaviour. Note this is true even though both algorithms return the same solution quality at the end of the solving period shown. As a secondary aim, we would like, if possible, to have an earlier termination time in addition to improved anytime behaviour.

Observe that `decomp` is an anytime algorithm in terms of robustness to uncertainty, i.e. the number of realisations covered by the conditional decision it computes. Answer quality increases with time because, given execution may commence at any point, it is better to have a conditional plan more likely to cover the realisation actually observed. Indeed, if allowed to run to termination, the algorithm produces an optimal conditional decision; if stopped earlier, the conditional decision covers a subset of the good realisations. In terms of plan execution, however, `decomp` fails to ensure that a valid plan is *always* available (the first part of an anytime property): if the observed realisation is not covered by the conditional decision at the time of interruption, the algorithm does not provide a valid initial control action.

Note that, fitting the assumptions of the control problem, the model of uncertainty in the SCPP is deterministic (i.e. an implicit uniform distribution over the parameter domains). This does not rule out algorithms based on sampling the parameter space. However, sampling may not lead to an immediately executable control decision on interruption; like `decomp`, such an approach may be anytime with respect to plan robustness but not plan executability.

### Environment Selection Heuristic

(Fargier, Lang, & Schiex 1996) note that heuristics may be used in line 5 of Algorithm 1, although none are proposed. The algorithm terminates when the set $E$ is empty. Every iteration through the main loop removes one environment $e$ from $E$. Judicious choice of $e$ may speed the termination or improve the anytime behaviour w.r.t. robustness, or both.

We propose five heuristics for environment selection:

- ***random***: pick the next environment at random. This is our default heuristic, used as a baseline to evaluate the others.
- ***most uncertainty***: pick the environment with the most uncertainty. That is, choose $e$ to maximize $\prod_{\lambda_i \in e} |L_i|$.
- ***least uncertainty***: pick the environment with the least uncertainty. That is, choose $e$ to minimize $\prod_{\lambda_i \in e} |L_i|$.
- ***most restricting***: pick the environment that most constrains the variables' domains. That is, for each $e$, impose the constraints $\mathcal{C}_e$ in line 6 of Algorithm 1, and compute $\prod_i |D_i|$. Choose $e$ to minimize this quantity.
- ***least restricting***: pick the environment that least constrains the variables' domains. I.e. impose the constraints $\mathcal{C}_e$, compute $\prod_i |D_i|$, and choose the maximizing $e$.

These heuristics are analogous to variable selection heuristics in finite domain CSP solving (Cheadle *et al.*

**Algorithm 2** Computation by incremental plan horizon

```
1: S ← ∅
2: for h = 1 to H do
3:    let S_h be output of decomp on horizon h automaton
4:    if decomp ran to completion then
5:       S ← S_h
6:    else
7:       {keep existing decisions for uncovered realisations}
8:       for each realisation covered by S_h do
9:          update S by S_h
10: return S
```

2003). Pursuing this link, we also considered a heuristic to pick the most or least constraining environment: that whose realised CSPs are the most or least constrained (precisely, maximize or minimize the sum of a constrainedness metric, summed over all the realised CSPs corresponding to realisations in the environment). However, preliminary experiments indicated such a heuristic has poor performance. This seems to be caused by a weak correlation between the constrainedness of the realised CSPs arising from an environment, and the difficulty of solving the whole mixed CSP. Thus we did not consider such a heuristic further.

### Incremental Horizon

The SCPP is naturally parameterized by the planning horizon, $H$. Running `decomp` to completion provides the sought optimal conditional plan. Interrupting the algorithm at any point provides a partial plan. As we have observed, since this plan is partial, in terms of execution it may not cover the realisation that actually occurs.

To better provide for plan execution, a second means of ensuring anytime behaviour is to iteratively plan for longer horizons, $h = 1, \ldots, H$. A new plan is generated from scratch at each iteration, avoiding any myopia, but at the cost of not reusing the previous solution. We permit the algorithm to be interrupted at the completion of any horizon. The resulting optimal conditional decision for horizon $h$ provides the initial steps of a complete plan for horizon $H$. We also permit `decomp` to be interrupted before completing a horizon. The plan for horizon $h$ then consists of the decisions for the covered realisations, together with, for the uncovered realisations, the decisions from horizon $h-1$.

More specifically, the time interval $[0 \ldots h],\ h \leq H$, defines a subproblem which is a subpart of the original SCPP instance. The subproblem is obtained by ignoring decision variables and parameters in the interval $[h+1, H]$, and relaxing associated constraints. Identifying these items to omit is straightforward due to the formulation of the model; omitting them yields a well-formed mixed CSP that describes the planning problem for the limited horizon $h$. The *incremental horizon* method starts from $h = 1$, and increments $h$ each time the subproblem is successfully solved. If interrupted, the method thus provides a plan up to time event $h-1$.

Algorithm 2 summarizes the method. As stated, conceptually it operates by solving incrementally larger subproblems. The advantage is that, in a given computation time, the plan produced may cover more of the good, possible realisations, compared to the plan produced by `decomp` for horizon $H$ in the same time. Indeed, suppose a plan for horizon $H$ is desired and computation time is limited to $T$ (which we do not assume is known to the algorithm). Running Algorithm 1 for time $T$ might give a conditional plan that covers $70\%$ of realisations, say. The conditional plan it yields is not optimal. Instead, running Algorithm 2 for the same time might give a plan that covers only $40\%$ of realisations with a horizon-$H$ decision, but all realisations are covered with some decision: say that for the horizon-$(H-1)$ decision. Thus we have an optimal conditional plan and, as the autonomous system begins plan execution, it can undertake further computation to extend the horizon-$(H-1)$ decisions to horizon-$H$.

In terms of execution, Algorithm 2 thus has the advantage over Algorithm 1 that a valid initial action is for certain available (once the problem is solved for horizon 1, which is expected rapid). Upon interruption, execution can proceed by checking whether the realisation observed is covered by the horizon $h$ decision. If not, the horizon-$(h-1)$ decision for it is used. This checking requires little computation.

The incremental horizon method is orthogonal to the environment selection heuristics. Any heuristic may be used in the invocation of `decomp` in line 3 of Algorithm 2. In the experimental results that follow, we hence evaluate the behaviour of incremental horizon both with the default *random* heuristic and with the others proposed above.

## Experimental Results

In this section we report an empirical assessment of the `decomp` algorithm on three SCPP instances. The aim of the experiments was to evaluate: (1) the impact of the environment selection heuristics on anytime behaviour with respect to robustness to uncertainty (measured by the completeness of the decision); and (2) the effectiveness of incremental horizon for producing anytime behaviour with respect to plan executability (measured by whether the decision contains an initial action for the actual realisation of the world).

The results reported were obtained on a 2GHz linux PC with 1GB of memory, using the constraint solver ECL$^i$PS$^e$ version 5.7 (Cheadle *et al.* 2003); timings are in ms. Table 1 summarizes the characteristics of the three SCPP instances. For each automaton, we considered three degrees of uncertainty: moderate, average and large, denoted A–C respectively. We also considered performance requirements between 20–80% (recall equation (2)). This gives two parameters for each problem instance.

We imposed a timeout on any single run of `decomp`, depending on the complexity of the automaton; the values are given in Table 1. Note the nonlinear nature of the constraints of Tracker means that solving for this automaton is markedly

| automaton | states per horizon | uncertainty per horizon A | B | C | timeout |
|---|---|---|---|---|---|
| AOCS | 5 | 2 | 4 | 5 | 200s |
| Thruster | 8 | 7 | 14 | 23 | 2000s |
| Tracker | 7 | 6 | 9 | 16 | 18000s |

Table 1: Characteristics of the benchmark problem instances

more difficult; hence the greater timeout permitted.

## Environment Selection Heuristic

We first consider the five environment selection heuristics. We measure quality by the number of good and possible realisations covered by the conditional decision, plus the number of bad realisations marked as bad, after a given computation time. That is, the quality is $Q_1(t) = |D| + |B|$, where $D$ and $B$ are as in the notation of Algorithm 1.

Figures 4(a)–4(f) show the quality (realisations covered) versus solving time (ms). Throughout, the vertical axis is shown on a log scale, i.e. $\log Q_1(t)$. Figure 4(a) shows the typical result for the AOCS instance: the best heuristic is *least uncertainty*, followed by *most restricting*; these are both better than *random*. The worst heuristic is *least restricting*; *most uncertainty* is slightly better.

Figures 4(b)–4(d) demonstrate the performance of the heuristics for Thruster is more varied. For most instances of uncertainty, performance, and horizon, *least uncertainty* is the best heuristic and *random* is second or third. However, for some instances, *least uncertainty* does not have maximal $Q_1(t)$ for all $t$. First look at Figures 4(c)–4(d). Note the scales of these two graphs are chosen to best compare the relative heuristic performances, not to show their overall anytime shape; hence some curves quickly reach the maximum quality shown, which is not the maximum attained. These graphs are for instances just before and just after infeasibility (which here occurs beyond horizon 6). In the former, *least uncertainty* is best at all times. In the latter, however, it is inferior to some other heuristics (in particular, to *random*) until about 2500ms, after which it strongly dominates. Heuristic *most restricting* exhibits poor behaviour.

Next look at the rare result in Figure 4(b). In this critically constrained problem, *random* is best at first, until overtaken by first *most uncertainty* then *least restricting*. Further, *least uncertainty* exhibits poor anytime performance. While exceptional, this instance indicates that no one heuristic always dominates. As in many algorithms that search through a plan or state space, the choice of heuristic is itself heuristic.

The results for Tracker confirm those for AOCS. Figures 4(e)–4(f) show *least uncertainty* as the best heuristic. Note it not only has a better performance profile, but also achieves much earlier termination than the other heuristics.

## Incremental Horizon

We now consider the incremental planning method. Here, we measure quality by the horizon attained after a given computation time. That is, the problem is solved incrementally for horizons 1, 2, ..., and the times $t_i$ recorded. The cumulative time for horizon $h$ is computed as $t_h = \sum_{i=1,\ldots,h} t_i$, and the quality is $Q_2(t) = \max\{h | t_h \leq t\}$.

Figures 5(a)–5(f) show log of the quality (horizon attained) versus solving time (ms). The shape of the curves indicate that Algorithm 2 provides acceptable anytime behaviour. However, performance strongly depends on the environment selection heuristic. Since incremental horizon is built on `decomp`, this might be expected.

Across the three automata, the performance of the *random* heuristic is broadly second or third of the five heuristics considered. For AOCS (Figures 5(a)–5(b)), the best heuristic is *least uncertainty*, followed by *most restricting*; these are both better than *random*. The worst heuristic is *least restricting*; *most uncertainty* is slightly better. The performance of *most restricting* declines beyond horizon 6; beyond this point, *random* has better performance.

For Thruster and Tracker (Figures 5(c)–5(f)), the results are similar. The best heuristic is *least uncertainty*, and overall *random* is next best. For the Tracker instance A 20% (Figure 5(e)), beyond horizon 4, the remaining three heuristics struggle; *most uncertainty* is the best of them. For B 40% (Figure 5(f)), *random* and *least restricting* dominate about equally. The results for Thruster (Figures 5(c)–5(d)), while similar, show strongly that poor heuristics for environment selection give very poor performance. This appears to be due to the large number of environments that must be maintained by Algorithm 1; the algorithm suffers from a lack of memory, and the timeout is reached for Algorithm 2 while it is still considering a low horizon $h$.

## Discussion

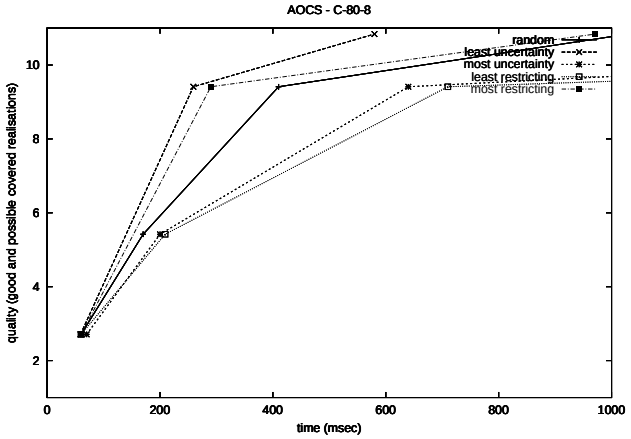
Of the environment selection heuristics, *least uncertainty* has the best overall performance, in terms of both metrics of quality. For the direct use of `decomp` (i.e. $Q_1(t)$), there are instances where other heuristics are better. In some instances, there is a 'cross-over' point (e.g. Figure 4(d)) prior to which another heuristic dominates, and after which *least uncertainty* dominates. For the incremental horizon use of `decomp` (i.e. $Q_2(t)$), *least uncertainty* dominates in almost all instances; we observe no cross-over behaviour.

We can make the analogy between *least uncertainty* (smallest environment first) and the *first fail* (smallest domain first) variable selection heuristic for classical CSP. 'First fail' is known as an often effective choice of variable selection heuristic (Cheadle *et al.* 2003). However, just as it is not the best heuristic for every CSP, so *least uncertainty* is not the best for every mixed CSP: Figure 4(b) shows a critically-constrained problem where the best heuristic is initially *random* then *most uncertainty*.

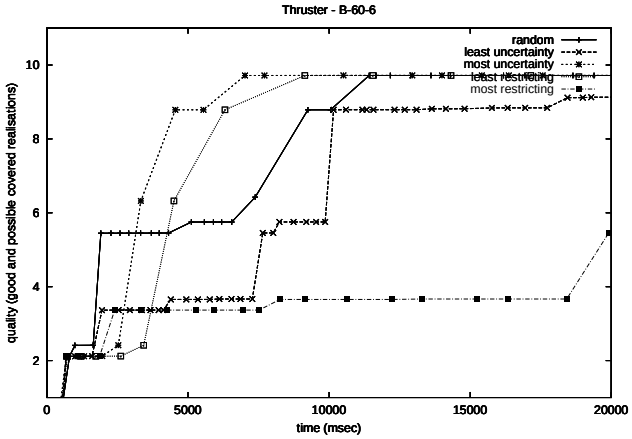
Secondly, overall *random* is consistently neither the best nor worst heuristic, as expected. On balance, its performance across the instances and across Algorithms 1 and 2 is second behind *least uncertainty*. Heuristics based on the size of variable domains (*most*/*least restricting*) vary in effectiveness between problem instances: e.g. *most restricting* is acceptable in Figure 4(a) but very poor in Figure 4(c). Note the size of variable domains loosely corresponds to the number of potentially feasible actions at a given plan state.

Thirdly, the results suggest that incremental horizon is effective in providing anytime behaviour with respect to plan executability, particularly for lesser horizons. When the subproblems becomes hard (e.g. from $h = 4$ for Thruster), the rate of increase of solution quality declines. This is more marked when the performance requirement is higher, perhaps a result of the problem then being over-constrained.

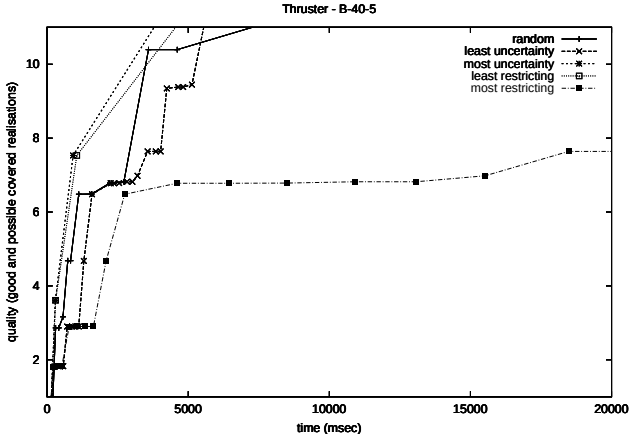
Since the SCPP is easy to solve for modest horizons, a possible approach might be: begin with Algorithm 2 and the *random* heuristic (which has no overhead to compute), and later switch to Algorithm 1 with the *least uncertainty* heuristic (the most effective overall). Further experimental work is needed to investigate this hybrid possibility.



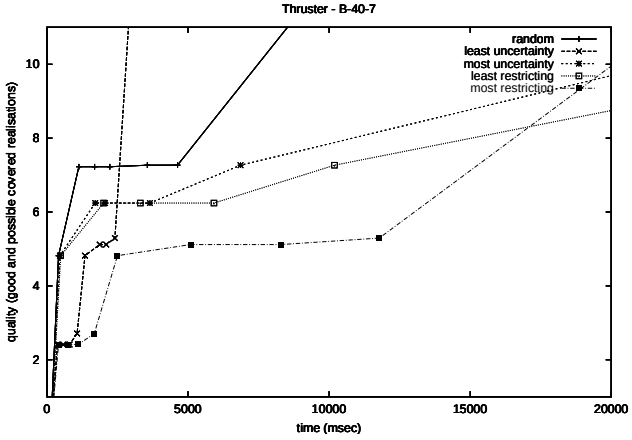

(a) AOCS C 80% horizon 8



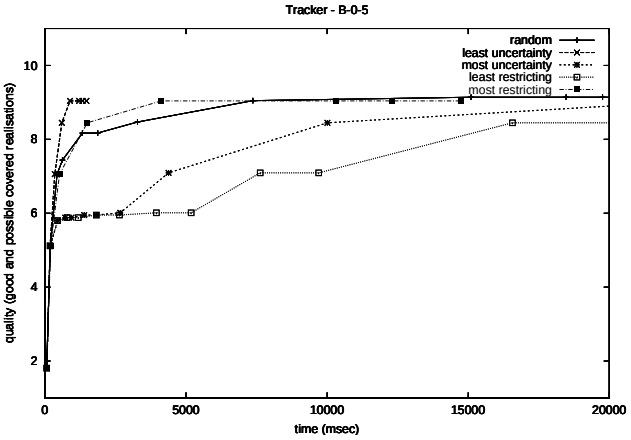

(b) Thruster B 60% horizon 6



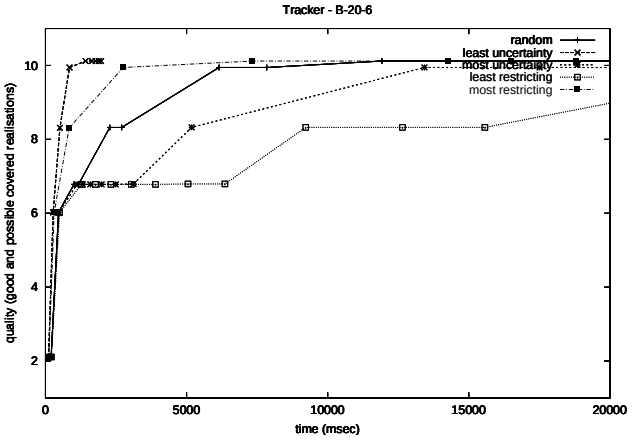

(c) Thruster C 40% horizon 5




(d) Thruster B 40% horizon 7

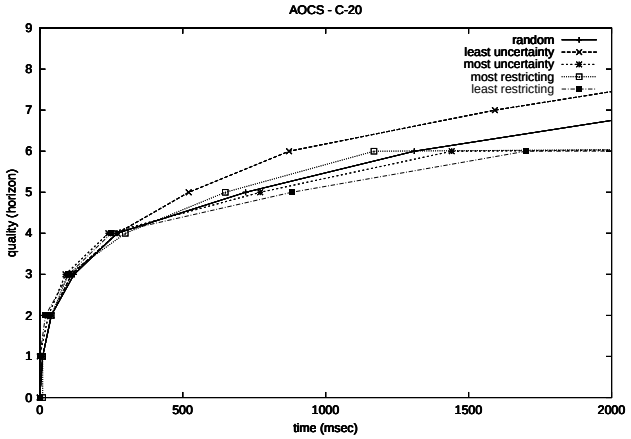



(e) Tracker B 0% horizon 5

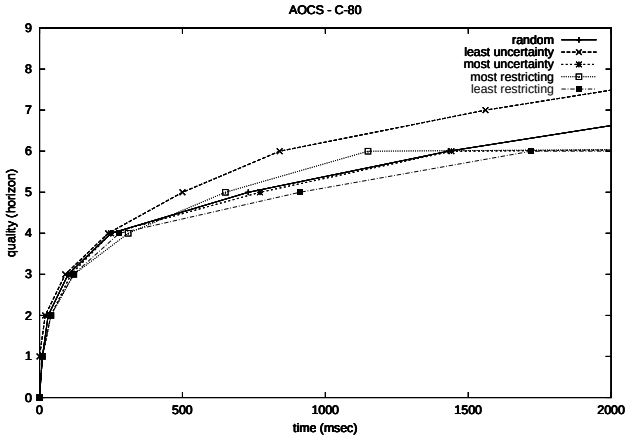



(f) Tracker B 20% horizon 6

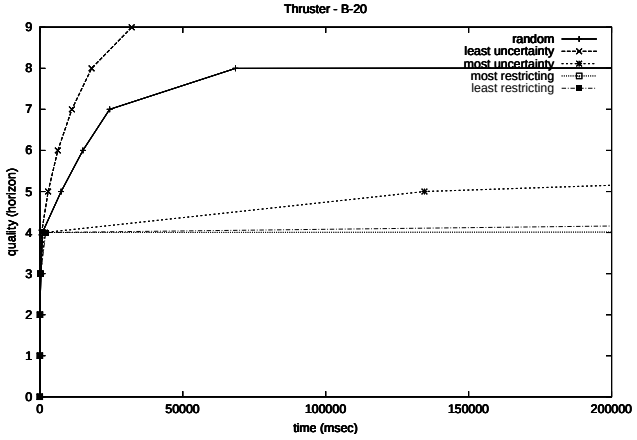
Figure 4: Anytime behaviour w.r.t. robustness of environment selection heuristics



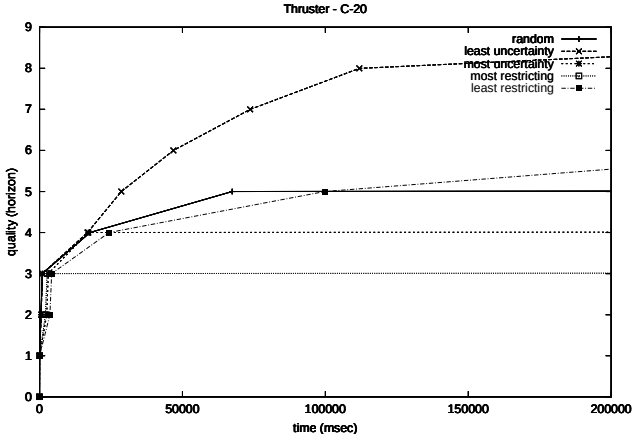

(a) AOCS C 20%



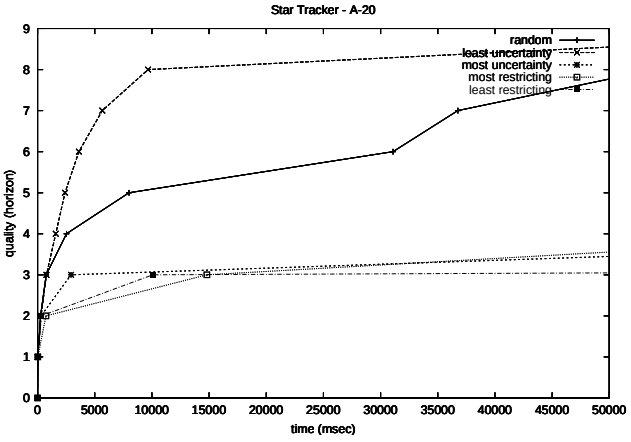

(b) AOCS C 80%



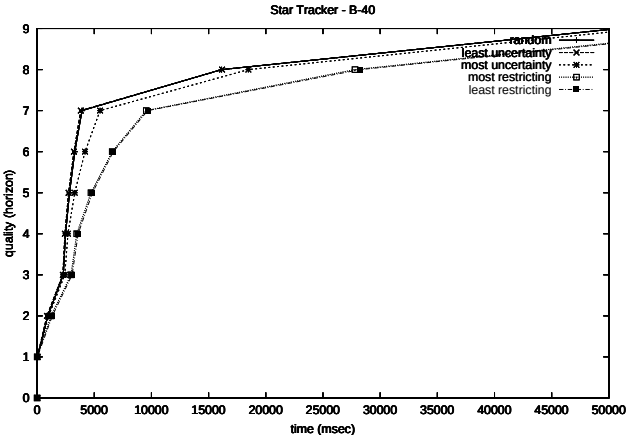

(c) Thruster B 20%




(d) Thruster C 20%




(e) Tracker A 20%




(f) Tracker B 40%

Figure 5: Anytime behaviour w.r.t. executability of incremental horizon

## Related Work

One approach to deal with uncertainty in planning is to continuously adapt a plan according to the changing operating context. Plan adaptation is performed upon an unexpected event, system failure or goal (mission) update. Response time can be reduced by using, for example, iterative repair techniques (Chien *et al.* 2000). Rather than reacting, our approach here is based on proactive but online anticipation of the environment or other changes. In exchange for the space overhead of storing multiple plans, this has the operational advantage of enabling the system to reason more globally and react more quickly. However, plan monitoring, and possibly reactive plan repair, are important to handle unanticipated contingencies such as hardware failure. Further, online adaptation of conditional plans can reduce the size of the plan to be stored (Bresina & Washington 2001).

In practice, plan management and execution often adopts a hybrid proactive and reactive form (Verfaillie 2001). This is true for one of the most comprehensive approaches to uncertainty in the aerospace domain, *Reactive Model Programming Language* (RMPL) (Williams *et al.* 2003). RMPL and the approach used in this paper are both model-based and share the use of constraint-based automata. However, whereas the SCPP considers the control of a single component in view of anticipated uncertainty, RMPL is a control approach for a combined system, with the focus on managing the complexity of component interaction.

Our work is motivated by problems situated at the interface of system engineering, planning and control theory. Application of much classical work on planning to aerospace component control is difficult, due to challenging domain-specific requirements (Jónsson *et al.* 2000). Further, actions must be scheduled with respect to rich temporal constraints, the system must cope with large-scale problems, and for low-level components, behaviour must be guaranteed in the worst case. The latter point, together with the difficulty of estimating probabilities, also hampers the use of Markov Decision Processes (MDPs).

Our approach follows (Boddy & Dean 1994) in constructing flexible online solutions in the form of conditional plans. (Castillo, Fdez-Olivares, & Gonzáez 2002) present a hybrid hierarchical and conditional planning approach for generating control of autonomous systems. Similar to our approach, during execution of the plan, a branch is selected according to the observed values of *runtime variables*. Our planning, however, is situated in an online context and also addresses time, resources, and uncertainty.

Similarities with several techniques in control theory exist mainly because environment assumptions apply to both the planning and control components of a system architecture, such as Figure 1, especially if the system must behave autonomously. However, the design of standard control components differs from our planning approach. In general, control techniques do not address the mission/operation level of granularity. Moreover, control properties like stability are expressed using an infinite horizon and a cost function defined over continuous domains.

Despite these differences, the vision of coexisting control and planning components in a unified system architecture motivates us to consider specific control techniques: hybrid control, model predictive control, and adaptive control (Levine 1996). These areas of control theory are applicable for autonomous systems evolving in uncertain and hostile environments. However, each only partially address the type of problems motivating this paper. The hybrid integration of planning, scheduling and control, supported by constraint solving techniques, is a potentially compelling avenue for the development of future autonomous systems.

An SCPP instance formulates a unique, rather than sequential, decision problem. Approaches to sequential decision making include MDPs and influence diagrams, and constraint-based frameworks, such as stochastic CSP (Manandhar, Tarim, & Walsh 2003), an extension of mixed CSP.

Lastly, anytime algorithms for classical CSPs have been built by considering a partial CSP and using branch-and-bound or local search (e.g. (Wallace & Freuder 1996)); (Cabon, de Givry, & Verfaillie 1998) address anytime computation of constraint violation lower bounds. For finding robust 'super' solutions, anytime algorithms have also been built with branch-and-bound (Hebrard, Hnich, & Walsh 2004). While anytime solving is related to incremental solving of CSPs, the focus there is on efficiently propagating the changes implied when a variable's domain changes.

## Conclusion and Future Work

Anytime behaviour is an important requirement for the aerospace domain. Motivated by an online planning problem for aerospace component control, this paper studied anytime planning under uncertainty with full observability mixed CSPs. We proposed two enhancements to the existing decomposition algorithm: heuristics for selecting the next environment to decompose, and solving of incrementally larger subproblems. Our empirical results indicate that incremental horizon planning provides effective anytime behaviour with respect to plan executability, and that it can be combined with the decomposition heuristics.

The heuristics we considered are applicable to solving any mixed CSP by the decomposition algorithm. Overall, the heuristic *least uncertainty*, which is analogous to 'first fail' for finite domain CSPs, gives the best performance. We have yet to consider heuristics based on analysis of priorities (e.g. criticality) from a model of system operation.

The incremental horizon method is specialized for the SCPP. By replacing the decomposition algorithm with an incremental version, we ensure anytime behaviour in terms of plan execution. However, the broader idea of decomposition into incremental subproblems, as a means of anytime solving, applies to any mixed CSP for which a suitable sequence of subproblems can be identified.

Incremental horizon is a baseline approach with similarities to iterative deepening. In future work, we want to complete our investigation by evaluating how often it produces plans for horizon $H$ based on partial plans for a lower horizon, as described earlier. Initial results indicate a trade-off according to the problem hardness: critically-constrained instances have fewer feasible actions (so greater overlap between subproblems) but are harder to solve (so fewer realisations covered in any given time). Because of the complicated mapping between high-level planning decisions and variables in the constraint model, we want to eval-

uate the methods considered here on other and larger SCPP instances (Yorke-Smith 2004), besides mixed CSP models arising from other planning problems.

Algorithm 2 produces a succession of limited horizon plans. In our current implementation, planning at horizon $h$ makes no reuse of the horizon $h-1$ plan. We thus want to explore solution reuse as a planning heuristic between different horizons, and to compare our proactive approach to reactive solving of the single horizon control problem at different times. Both reactive and limited horizon proactive planning may lack completeness. In particular, if in Algorithm 2 we base planning for the next horizon on the solution for the last, the plan generated to horizon $h-1$ may not be extendible to horizon $h$: for example, if it uses a resource (such as energy) that any plan for the next horizon may need. Thus this idea is more suited to managing a system whose global return must be optimized (for example, an observation system that must make the most useful observations over all its life) rather than when managing a system in order to lead it to a designated goal state.

The 'cross-over' between different heuristics over time suggest that meta-reasoning on the solving algorithm may yield the best anytime behaviour in practice. More generally, this reasoning can take into consideration (Hansen & Zilberstein 1996): the current state of the plan (such as what percentage of realisations it presently covers); the expected computation time remaining, if an estimate is available; the cost of computing the different heuristics; and the opportunity of switching between algorithms during solving. Such reasoning can be consolidated by analyzing the complexity of decomposition algorithm, thus providing metrics to understand how problem difficulty is balanced between the set of realisations and the core planning problem.

Driven by our motivational problem, in this paper we have considered only the full observability case, an instance of contingent planning; an interesting direction would be to consider anytime solving in the no observability case. Here, the outcome sought to a mixed CSP is a single robust solution that covers as many realisations as possible, i.e. a conformant plan. As such, there are links to anytime methods for robust solutions to CSPs and to solving mixed CSPs with probability distributions over the parameters, and to probabilistic planning (e.g. (Onder & Pollack 1999)). In particular, now scenario sampling methods for stochastic CSPs give the opportunity for an anytime algorithm (Manandhar, Tarim, & Walsh 2003).

**Acknowledgments.** We thank K. Myers, T. Winterer, and the anonymous referees for helpful comments, and the participants of the Changes'04 Workshop for suggestions on an earlier version. This work was undertaken when the second author was at IC–Parc, partially supported by the EPSRC under grant GR/N64373/01.

## References

Allo, B.; Guettier, C.; Legendre, V.; Poncet, J.; and Strady-Lecubin, N. 2002. Constraint model-based planning and scheduling with multiple resources and complex collaboration schema. In *Proc. of AIPS'02*.

Arkin, R. C. 1998. *Behavior-Based Robotics*. Cambridge, MA: MIT Press.

Bertoli, P., and Cimatti, A. 2002. Improving heuristics for planning as search in belief space. In *Proc. of AIPS'02*, 143–152.

Boddy, M. S., and Dean, T. L. 1989. Solving time-dependent planning problems. In *Proc. of IJCAI'89*, 979–984.

Boddy, M., and Dean, T. L. 1994. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence* 67(2):245–285.

Bresina, J. L., and Washington, R. 2001. Robustness via run-time adaptation of contingent plans. In *Proc. of AAAI 2001 Spring Symposium on Robust Autonomy*.

Cabon, B.; de Givry, S.; and Verfaillie, G. 1998. Anytime lower bounds for constraint violation minimization problems. In *Proc. of CP'98*, LNCS 1520, 117–131.

Castillo, L.; Fdez-Olivares, J.; and Gonzáez, A. 2002. Shifting AI planning technology from automated manufacturing to autonomous operation and control in space missions. In *Proc. of AI Planning and Scheduling For Autonomy in Space Applications*.

Cheadle, A. M.; Harvey, W.; Sadler, A. J.; Schimpf, J.; Shen, K.; and Wallace, M. G. 2003. ECLiPSe: An Introduction. Technical Report IC-Parc-03-1, IC–Parc, Imperial College London.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve responsiveness of planning and scheduling. In *Proc. of AIPS'00*.

Fargier, H.; Lang, J.; Martin-Clouaire, R.; and Schiex, T. 1995. A constraint satisfaction framework for decision under uncertainty. In *Proc. of UAI'95*, 167–174.

Fargier, H.; Lang, J.; and Schiex, T. 1996. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proc. of AAAI-96*, 175–180.

Frank, J., and Jónsson, A. 2004. Constraint-based attribute and interval planning. *Constraints* 8(4):339–364.

Hansen, E. A., and Zilberstein, S. 1996. Monitoring the progress of anytime problem-solving. In *Proc. of AAAI-96*.

Hebrard, E.; Hnich, B.; and Walsh, T. 2004. Super solutions in constraint programming. In *Proc. of CP-AI-OR'04*, 157–172.

Jónsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in interplanetary space: Theory and practice. In *Proc. of AIPS-00*.

Levine, W. S., ed. 1996. *The Control Handbook*. CRC Press.

Manandhar, S.; Tarim, A.; and Walsh, T. 2003. Scenaio-based stochastic constraint programming. In *Proc. of IJCAI'03*.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103(1–2):5–48.

Onder, N., and Pollack, M. E. 1999. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proc. of AAAI-99*.

Verfaillie, G. 2001. What kind of planning and scheduling tools for the future autonomous spacecraft? In *Proc. of the ESA Workshop on On-Board Autonomy*.

Wallace, R. J., and Freuder, E. C. 1996. Anytime algorithms for constraint satisfaction and SAT problems. *SIGART Bulletin* 7(2).

Williams, B. C.; Ingham, M.; Chung, S. H.; and Elliott, P. H. 2003. Model-based programming of intelligent embedded systems and robotic explorers. *IEEE Proceedings* 9(1):212–237.

Yorke-Smith, N., and Guettier, C. 2003. Towards automatic robust planning for the discrete commanding of aerospace equipment. In *Proc. of 18th IEEE Intl. Symposium on Intelligent Control (ISIC'03)*, 328–333.

Yorke-Smith, N. 2004. *Reliable Constraint Reasoning with Uncertain Data*. Ph.D. Dissertation, Imperial College London.

# On-line Mission Planning for Autonomous Vehicles

**E. Chanthery** and **M. Barbier** and **J-L Farges**
Office National d'Etudes et de Recherches Aerospatiales (ONERA)
Systems Control and Flight Dynamics Department (DCSD)
BP 4025 - 2, av. Edouard Belin - 31055 Toulouse CEDEX 4 - FRANCE
(Elodie.Chanthery, Magali.Barbier, Jean-Loup.Farges)@onera.fr

**Abstract**

In mission planning problems applied to autonomous vehicles, the plan is relative to a vehicle motion and has a not *a priori* fixed number of objectives to fulfil. Here, the problem is formalized by the association of an utility for each objective and a cost relative to time and resources consumption for the actions. The utility of an objective is a function not only of the importance of the objective, but also of time and resources. The resources consumption depends on the actions of the vehicle and on the environment. The criterion of choice between all the possible paths is the difference between costs and rewards: the criterion is a non-monotonic function.

For the resulting domain, a planning algorithm template is described together with a set of solving subcomponents: four cost evaluation methods, two pruning methods and four arrangement methods. The approach is applied to an aerial autonomous vehicle performing a military observation mission. Experiments, based on this application, show that for a replanning scenario ordered best-first search fails in finding a solution and that cost evaluation methods based on the relaxation of resources constraints present poor performances. At the opposite, best-first search methods associated with heuristics taking into account resources may be applied on-line.

## Introduction

The autonomy of a vehicle is characterized by the interaction between the vehicle and the operator: the more abstract the operator decisions are, the more autonomous the vehicle is. Missions with limited communications between the vehicle and the operators require some decisional autonomy. Indeed, the vehicle has not only to follow the current plan, but also to react to events occurring during the mission. Two main approaches exist to obtain this behavior: off-line and on-line planning.

For off-line planning, the easiest solution would be to use proactive planning, which avoids all the risks that might occur during the mission. However, this solution is not applicable for most of the realistic problems. Indeed, in many situations risks can be inhibited only by actions that are not compatible with the goals of the mission. For instance in a military mission, motion actions that avoid the enemy area inhibit risks due to threatening events from the enemy, while the vehicle has to enter the enemy area in order to achieve the goal. Another solution is to use conditional or probabilistic planning. In this case, the plan is a tree of actions. During the execution, a specific path in the tree is followed depending on actually observed events. Highly conditional plans are obtained solving Markov Decision Processes or Partially Observable Markov Decision Processes. The off-line computation of the solution of the Bellman's equation provides, for each actual or belief state, the action to carry on. This approach is studied for instance for exploration planning (Teichteil & Fabiani 2004) and in the context of high level decision making (Schesvold *et al.* 2003). However, solutions are intractable except for small problems. Only problems with few states and actions can be solved exactly. Moreover, a probabilistic model of instants of occurrence of the events and of their types has to be defined.

For on-line planning, the plan is defined as a sequence of actions. During the execution of the mission, three approaches can be used. The real-time planning or continuous planning aims at planning actions taking into account the environment updates. Algorithm $CD^*$ (Stentz 2002) replans an optimal solution with global constraints in less than one second. The algorithm works on a graph, where the arcs are re-evaluated. Replanning function re-computes only the arcs, whose value changed. However, the algorithm is an $A^*$ (Nilson 1971) if the number of arcs, whose value changes, are significant. Anytime planning aims at always improving the current plan. The plan computation stops when the instant for executing the first possible action is reached. This approach is applied for instance for autonomous earth watching satellites (Damiani, Verfaillie, & Charmeau 2004). Finally, the third idea is to link the plan computation with the occurrence of events. The plan computation starts when an event occurs and stops when the instant for executing the first possible action is reached. This approach is applied for instance for an Autonomous Underwater Vehicle (Barbier, Lemaire, & Toumelin 2001). It is the most adapted to an on-line mission planning for autonomous vehicles.

The purpose of this work is to address a large class of on-line mission planning problems. After a brief overview of the problem and related works, the paper introduces a for-

malism based on the world abstraction concept. A two-level hierarchy is used in order to organize the description of the problem. The low level deals with the different spatial and temporal ways to achieve each goal. Then, a section is devoted to a basic planning algorithm and several alternatives. The experiments, presented in the last section, assess the performances of the proposed alternatives.

## Mission Planning Problem

### Problem definition

A mission is made up of actions: motion actions, actions on the environment, information-gathering actions... Resources spent while carrying out actions are available in limited quantity. For a large family of problems, resources are consumable. A mission has a beginning and an end between which all the actions are carried out. It also has an object, which is a set of objectives to fulfil.

There are different ways that can be used to fulfil each objective. Each way corresponds to the choice and the sequencing of a set of actions. The goal of the mission planning is to select the objectives to fulfil and to find the way of fulfilling them. In this work, the solution optimizes a criterion that takes into account utilities for each objective while meeting time and resource constraints. The utilities and constraints are nonlinear functions of time and resources at the various instants when the actions that lead to the achievement of the objectives are carried out. The criterion is so a non-monotonic function of time and resources.

### Related works

In the literature, three main approaches that find a plan that can be applied for solving the mission planning problem can be encountered: motion planning, operational research graph optimization and classical artificial intelligence planning.

The goal of motion planning or path planning is to find the best path from the current vehicle position to a given position while avoiding obstacles, controlling its basic movements and the movements of its actuators. This planning approach addresses the optimization of a criterion based on resource consumption and can take into account time and resource constraints. However, motion planning does not address the choice of objectives to fulfil.

The operational research graph optimization approach addresses optimization of a criterion associated to objectives, as for instance the Travelling Salesman Problem. It sometimes deals with a constraint on the use of a single resource or on time. Utilities are constant and associated to a single node. The constraint on time or resource is linear.

Artificial intelligence approach tends to describe the planning problem in terms of actions and predicates, as STRIPS-like plannings (Fikes & Nilsson 1971). This type of planning presents a good flexibility for describing the different ways an objective can be achieved, but usually does not address the optimization of a non-monotonic criterion nor continuous resources consumption. Some of the AI planners are FF (Hoffmann & Nebel 2001), LPG (Gerevini, Saetti, & Serina 2004) or SGPlan (Wah & Chen 2004). Classical planning is usually done off-line. The generated plan is then fed to the on-line execution module.

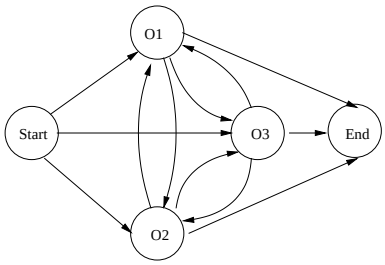



Figure 1: Operational research graph optimization approach: start, end of treatment and reward obtaining are simultaneous

In contrast to common AI planning descriptions, the mission planning problem has a set of goals with variable utilities and has a not *a priori* fixed number of objectives to fulfil.

Examples of planning for this sort of problem include many of NASA planning problems such as planning for telescopes like Hubble or SOFIA (Frank & Kurklu 2003). Existing planning systems, where goals are assumed to have uniform utility and define a conjunctive set, can not solve this type of problems. Three approaches are can be applied. Greedy approaches pre-select a subset of goals according to their estimated utilities and solve the problem for those goals. These methods, though efficient, can produce plans of low quality. Recently, Smith (2004) proposed a more sophisticated approach consisting in heuristically selecting a subset of goals that appear to be the most beneficial and then by using the classical planning search to find the solution of lower cost that achieves all selected goals. The second solution consists in transforming the problem into an integer programming problem (van den Briel *et al.* 2004). The third solution considers the rewards associated to the objectives and selects them during a search in a tree of actions having a cost.

### Problem difficulty

The objective achievement is seldom instantaneous: there are a start and an end of treatment and a moment when the reward associated with the objective is obtained. For example in an observation mission, an objective may be to scan an area to collect information. The scanning has a start and an end. Information collected is then sent to the ground-station when it is possible. For many problems of the graph optimization literature, these three events are simultaneous (Figure 1). For real problems of mission planning, this distinction is necessary (Figure 2). The distinction implies that the achievements of different objectives can be interlaced (Figure 3).

Moreover, each action introduced into the plan can be carried on in various ways, leading to different values for utilities, time and resource consumption. In this work, the solution optimizes a criterion that takes into account utilities for each objective and costs for achieving objectives.

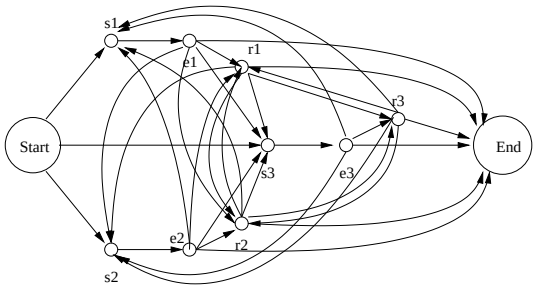



Figure 2: Mission graph: start ($s_i$), end ($e_i$) of treatment and reward obtaining ($r_i$) are distinct

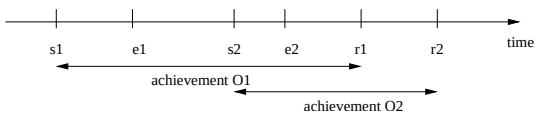



Figure 3: Interlacing of objectives achievement

## Proposed Formalism

The solution proposed here tends to unify and generalize some existing models related to mission planning problems in order to solve a large class of problems.

The idea is to use the notion of world abstraction (Galindo, Fernández-Madrigal, & González 2004) with a two-level hierarchy. A group of nodes of the lowest hierarchical level is abstracted to a single node at the highest hierarchical level, which becomes their supernode. Analogously, a group of arcs can be represented by a single arc at the highest level. In the literature, planners use abstraction to reduce computational cost. They first solve a problem in a simpler abstract domain and then refine the abstract solution, inserting details that were ignored in the most abstract domain. Here, the hierarchy is used in order to organize the description of the problem and may be used to guide the search. The highest level is a finite state machine where some vertices are the steps of the goal achievements and the transition arcs are abstract actions carried out. The state changes of the machine are due to the planned actions.

The abstract level plan thus contains a succession of states. It minimizes a criterion that takes into account the costs for carrying out the actions and the rewards obtained by the goal achievements. The criterion is a function of time and resources.

The finite state machine is thus supplemented by the lowest hierarchical level, which describes the state of the vehicle in time and geometrical space.

For example, for the mission graph on Figure 2, the vertices of the finite state machine are the steps {*Start*, $s_i$, $e_i$, $r_i$, *End*}. The transition arcs are the actions carried out: for example between *Start* and $s_i$, it could be a motion action, between $s_i$ and $e_i$, it could be an information-gathering action... The result of the planning contains a succession of states that may be *Start*-$s_2$-$e_2$-$s_1$-$e_1$-$r_2$-$r_1$-*End*. This level is the most abstract one. The finite state machine is supplemented by a less abstract level that depends on the application. For example in robotic problems, the finite state machine is supplemented with a level that indicates the speed and the exact location of the robot.

### High level description: objective achievement

Let consider $N$, a set of nodes. Let $W = \{W_1, \ldots, W_e\}$ be a partition of $N$. A relation $S$ is defined as follows: for a subset $W_i$, $S(W_i)$ is defined as the set $\{\ldots, W_j, \ldots\}$ of the subsets of the successors of $W_i$. $W_j$ is a successor of $W_i$ if there exists at least one possible high level action between $W_i$ and $W_j$.

A finite state machine is defined by $W$ and the relation $S$. Without a loss of generality:

- $W_1$ corresponds to the initial state.
- $W_e$ corresponds to the end of the mission.
- The transition function is defined by the transition possibilities (described by $S$) and the result of the planning (the best action defined by the plan).

For all $W_i$ in $W$, $W_i$ is reachable from $W_1$ and $W_e$ is reachable from $W_i$.

Let $P$ be the set of objectives to fulfil. Each objective $o$ in $P$ is defined by $(W_{s(o)}; W_{e(o)}; W_{r(o)})$ with $s(o), e(o)$ and $r(o)$ in $\{1, \ldots, e\}$. The objective is said to be *treated* if there is a transition from $W_{s(o)}$ to $W_{e(o)}$ and the associated reward is obtained when the finite state machine is in a state $W_{r(o)}$: the objective is said to be *achieved*. The reward is a real valued quantity (income). For each objective $o$ the finite state machine can be in the state $W_{s(o)}$ only once. An example of high level description is presented on Figure 5.

### Time and resources: justification for a low level description

Let $t_k$ and $r_k$ respectively be the instant and the vector of resources at state $W_k$; then the reward $R_o$ for achieving an objective $o$ is:

$$R_o(t_{s(o)}, r_{s(o)}, t_{e(o)}, r_{e(o)}, t_{r(o)}, r_{r(o)})$$

It is assumed that the function $R_o$ is bounded over the set of realistic input values.

At the end of the plan, the vehicle reaches $W_e$ with resources $r_e$. The cost function of the resources $R_e(r_e)$ is supposed to be decreasing with resources.

The planning goal is to find a sequence $Q$ of states $W_{\pi(1)}, \ldots, W_{\pi(q)}$ such that $\pi$ is a function from $\{1, \ldots, q\}$ to $\{1, \ldots, e\}$ with:

$$\pi(1) = 1; \quad \pi(q) = e; \quad W_{\pi(i+1)} \in S(W_{\pi(i)})$$

$Q$ has to minimize the difference $J$ between costs and rewards and to satisfy the constraints on resources. Let $E_o$ be the set of achieved objectives. Then:

$$J = R_e(r_e) - \sum_{o \in E_o} R_o$$

Resources constraints $C_e$ are given by the inequality $C_e(r_e) \geq 0$. Resources have the following property: for

a resource that decreases from one task to the next one, even if the variation of the resource between two tasks is not completely known, the relative cost of the current resource can be considered as already consumed. Knowing this, considering the state $W_{\pi(i)}$ of $Q$, the partial cost $R_e(r_{\pi(i)})$ on a current itinerary can be defined by:

$$R_e(r_e) = R_e(r_{\pi(i)}) + \sigma_{\pi(i),e} \text{ with } \sigma_{\pi(i),e} \in \mathbb{R}^+$$

The goal achievement problem is not fully specified because the evaluation of instants and resources depends on the choice of the node of $N$ in each $W_{\pi(i)}$ of the sequence and for each node of an instant to go. It has to be supplemented with a low level description.

### Low level description

The relation $S$ is expanded to the nodes level: given $n$ and $m$ in $N$, there exists one or several arcs from $n$ to $m$ if and only if there exist $i$ and $j$ with $n \in W_i$, $m \in W_j$ and $W_j \in S(W_i)$.

Let $A$ be the set of possible low level actions. For each couple ($W_i$, $W_j$) such that $W_j \in S(W_i)$, there exists a subset $A_{i,j}$ of $A$ indicating authorized actions between $W_i$ and $W_j$. Given $a \in A_{i,j}$, $m$ may be specialized in $m^a$, indicating that $m$ is reached with the action $a$.

For two chosen nodes $n_k$ and $n_{k+1}$ respectively in $W_{\pi(k)}$ and $W_{\pi(k+1)}$ of the sequence $Q$, if $n_{k+1}$ is reached by the action $a_{k+1}$, instants at $n_k$ and $n_{k+1}$ are linked by the relation:

$$t_{\pi(k+1)} = t_{\pi(k)} + \Delta^{a_{k+1}}_{n_k,n_{k+1}}$$

where $\Delta^{a_{k+1}}_{n_k,n_{k+1}}$ is bounded according to $n_k$, $n_{k+1}$ and $a_{k+1}$. The date of the beginning of the mission is known. Each node may have a time window.

The state variables are then the position of the vehicle, the state of the objectives (not treated, treated, achieved) and the levels of the resources.

### Resources consumption

Resources are supposed to be consumable:

$$r_{\pi(i+1)} \leq r_{\pi(i)}$$

Given a sequence of nodes $n_1, \dots n_i$ such that $n_k \in W_{\pi(k)}$, given a sequence of actions $a_2, \dots, a_i$ such that $a_k$ is the action between $n_{k-1}$ and $n_k$, the level $r^s_{\pi(i)}$ of the resource $s$ at a node $n_i$ is of the form:

$$r^s_{\pi(i)} = f^s_{\pi(i)}(n_1, \dots, n_i, a_2, \dots, a_i, \Delta^{a_2}_{n_1,n_2}, \dots, \Delta^{a_i}_{n_{i-1},n_i})$$

Two kinds of resources are considered: arc decomposable resources and global resources. For example, the mass of the vehicle at the current time is an arc decomposable resource, while the probability to be alive at current time is a global resource. For arc decomposable resources only, the level of the resource at a node $n_i$ may be expressed as:

$$r^s_{\pi(i)} = r^s_{\pi(i-1)} + \tilde{f}^s_{\pi(i)}(n_{i-1}, n_i, a_i, \Delta^{a_i}_{n_{i-1},n_i})$$

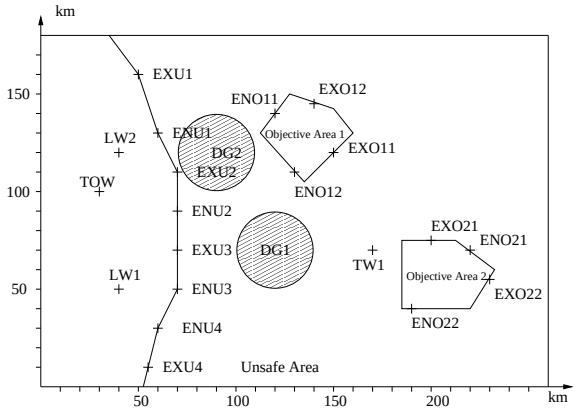



Figure 4: Mission map example

## Applicative Example

The formalism is illustrated on a military observation mission carried on by an autonomous unmanned aerial vehicle (Chanthery, Barbier, & Farges 2004). The environment is three dimensional, dynamic, uncertain and dangerous. It includes an unsafe area where the vehicle carries out operations, which are the objectives of the mission. The mission constraints are due to the objectives, the environment and the vehicle. The planning function has to select and order the best subset of objectives and to determine the arrival date at each waypoint, maximizing observation rewards and minimizing criteria on danger, fuel consumption and durations, while meeting the mission constraints. An example of a mission map including two objective areas is shown on Figure 4.

### High level description

The set $N$ of the nodes corresponds to the points of the geometrical space. It includes the take-off waypoint $TOW$, the set of landing waypoints $LW$, the set of entrance points of the unsafe area $ENU$, the set of exit points of the unsafe area $EXU$, for each objective, the sets of entrance $ENO_o$ and exit $EXO_o$ points and specific waypoints $\{TW_1, \dots\}$ for data transmission.

The partition $W$ is defined according to the type of each waypoint. Figure 5 illustrates the finite state machine corresponding to a planning beginning at the take-off point. $W_s$ is defined by the set containing $TOW$ for the mission preparation, or by the current physical position of the vehicle for reactive planning. $W_e$, the end of the plan, corresponds to $LW$. The transition function $S$ is defined by one or more possible trajectories between two sets of waypoints.

$P$ is composed of objective $O1$ and objective $O2$. $O1$ is defined by ($W_3$; $W_4$; $W_4$). Indeed, the data transmission for this objective is done at the exit point of the area. $O2$ is defined by ($W_5$; $W_6$; $W_7$). The data transmission for this objective is done at a transmission point because the objective area is out of range of the ground station.

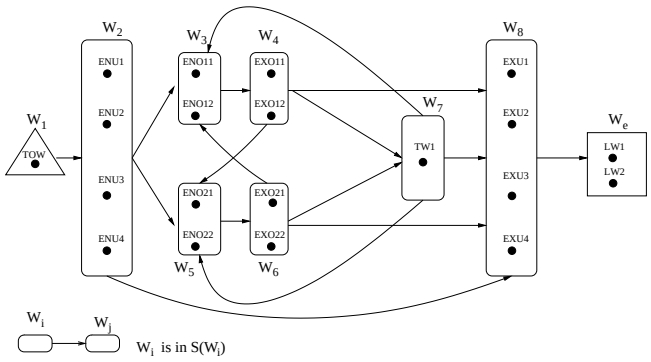



Figure 5: Finite state machine of the mission - Objective 1: $s(1) = 3, e(1) = 4, r(1) = 4$; Objective 2: $s(2) = 5, e(2) = 6, r(2) = 7$

### Time and resources

The resources vector $r \in \mathbb{R}^2$ contains the probability of being alive at current time (first component noted $r^1$) and the mass of the vehicle at current time (second component noted $r^2$). For a more complex description of the expression of these terms, see (Chanthery, Barbier, & Farges 2004). For each objective, the reward $R_o$ is defined by the function $G_o.p_o(t_{s(o)}).r^1_{r(o)}$ where $G_o$ is the maximum reward associated to $o$, $p_o(t_{s(o)})$ represents the quality of the observation at time $t_{s(o)}$ and $r^1_{r(o)}$ represents the probability of being alive at data transmission time. The cost function $R_e$ corresponds to the costs of danger and consumption:

$$R_e(r_e) = (r_1 - r_e)^\top.\mathbf{C}$$

where $\mathbf{C}$ is a vector of $\mathbb{R}^2$, whose first component is the price of the aerial autonomous system (vehicle and payload included), and whose second component is the price of the fuel per mass unit. So, costs are decreasing with resources.

The planning goal is to find a sequence of states beginning by the take-off waypoint, ending by the set of landing waypoints of the mission and using the possible trajectories between two sets of waypoints. The sequence has to minimize the difference $J$ between costs of danger and consumption and rewards obtained for the data transmission while satisfying the constraints on danger and fuel.

$$J = R_e(r_e) - \sum_{o \in E_o} G_o.p_o(t_{s(o)}).r^1_{r(o)}$$

The constraint $C_e(r_e) \geq 0$ expresses the fact that the vehicle has enough chances to finish its mission. It has the following form:

$$r_e - r_{min} \geq 0$$

Indeed, the probability of being alive at the end of the mission must be greater than a given limit ($r^1_{min}$) under which the vehicle is considered as destroyed. The fuel being limited, the mass of the vehicle cannot be lower than the mass without fuel $r^2_{min}$.

### Low level description

Different motion actions are possible to reach a node of $N$. If there is no danger, the motion action is the straight line. If there is a danger, it is possible to bypass the danger or to cross it. During the treatment of an objective, the vehicle can follow an outline or a trajectory for the area survey. Bounds on $\Delta^{a_{k+1}}_{n_k,n_{k+1}}$ are computed by considering on the one hand aerodynamic and propulsion characteristics of the vehicle and on the other hand the traveled distance, the average slope and the average height from the node $n_k$ to the node $n_{k+1}$ using action $a_{k+1}$.

Some nodes of $N$ have a time window. For the entrance and exit points of the unsafe area, time windows correspond to operational procedures to safely cross the frontier. For each objective, time window indicates the times when the observation is valid.

### Resources consumption

Resources are consumable, so they decrease with time. Fuel resource is decomposable. Let us simplify the notation $\Delta^{a_i}_{n_{i-1},n_i}$ in $\Delta$. The decrease of the fuel on the arc from node $n_{i-1}$ to node $n_i$ corresponding to the action $a_i$ is given by:

$$\tilde{f}^2_{\pi(i)}(n_{i-1}, n_i, a_i, \Delta) = \\ -\Big(\alpha(n_{i-1}, n_i, a_i)\frac{1}{\Delta^2} + \beta(n_{i-1}, n_i, a_i).\Delta^4\Big)$$

where $\alpha(n_{i-1}, n_i, a_i)$ and $\beta(n_{i-1}, n_i, a_i)$ are computed by considering the same parameters as for bounds on $\Delta^{a_i}_{n_{i-1},n_i}$.

On the contrary, the probability of being alive is not decomposable. It depends on the entire past path of the vehicle. Indeed, the probability of being alive is the product, on all the exposures to danger along the path, of the probability of surviving the considered exposure. It is given by:

$$f^1_{\pi(i)}(n_1, \ldots, n_i, a_1, \ldots, a_i, \Delta^{a_2}_{n_1,n_2}, \ldots, \Delta^{a_i}_{n_{i-1},n_i}) = \\ \prod_{m \in S_M} \prod_{e_m \in E_m} \Big(1 - \gamma_m.pt\big(\sum_{(n_{j-1},n_j,a_j) \in E_{e_m}} \Delta.\delta(n_{j-1}, n_j, a_j)\big)\Big)$$

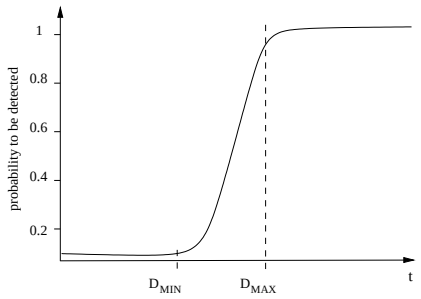
where $S_M$ is the set of threats, $E_m$ the set of exposures for threat $m$, $\gamma_m$ the probability that the threat $m$ actually exists and is able to destroy the vehicle when it is detected, $E_{e_m}$ the set of arcs exposed to the threat during exposure $e_m$, $\delta(n_{j-1}, n_j, a_j)$ the ratio of the arc $(n_{j-1}, n_j, a_j)$ that is exposed to the threat and $pt$ the probability of being detected in function of the time of exposure. The probability $pt$ is given on Figure 6.

The probability of being alive is not a linear function of exposure duration.

## Planning Algorithms

### Algorithmic framework

The plan search is performed on the tree of possible actions. Proposed algorithms are different from the ones of the literature: for each developed node, the precise evaluation of the criterion requires an optimization of the instants at each




$D_{MIN}$ : duration under which the vehicle has less than 1 % of risk to be detected by the threat

$D_{MAX}$ : duration above which the vehicle has more than 99 % of risk to be detected by the threat

Figure 6: probability of vehicle detection

node for the whole plan. The output is a sequence defined by an ordered list of nodes, each node being specialized by an action, and a vector of optimized durations between each pair of nodes. The planning algorithm is adapted for on-line replanning and so is able to begin at any node taking into account the updated situation. For each developed node, an optimization sub-problem is solved. Let $I$ be the partial criterion from $n_1$ to $n_i$:

$$I = R_e(r_{\pi(i)}) - \sum_{o \in E_o^i} R_o(t_{s(o)}, r_{s(o)}, t_{e(o)}, r_{e(o)}, t_{r(o)}, r_{r(o)})$$

where $E_o^i$ is the set of objectives achieved before $n_i$.

The goal is to optimize the partial criterion $I$ with respect to $\Delta_{1,2}^{a_2}, \ldots, \Delta_{n_{i-1},n_i}^{a_i}$ and $t_{\pi 2}, \ldots, t_{\pi i}$, under:

- the linear time constraints : equality constraints between $t_{\pi(k+1)}$, $t_{\pi(k)}$ and $\Delta_{n_k,n_{k+1}}^{a_{k+1}}$, inequality constraints on $t_{\pi(k)}$;
- the non linear constraints: $C_e(r_{\pi(i)}) \geq 0$.

The sub-problem is transformed into the optimization of a nonlinear criterion under linear constraints. It is solved by the Frank-Wolfe algorithm (Frank & Wolfe 1956). The computation of a first admissible sequence is useful for the reactive behavior of the system and for an efficient pruning of the tree. A first sequence is so searched without optimizing the durations and by developing a limited number of nodes. This search uses the same cost-evaluation, arrangement and pruning methods as the algorithm. Instants are then optimized for this sequence, given a bounded value for the criterion used by the algorithm. If the optimization is impossible, no first sequence is found and the bound value remains infinite.

Let us define some notations: $n_1$ is the node of the planning beginning; $BOUND$ is the current optimal value of the criterion for a sequence from $n_1$ to an end node, with an infinite initial value; $P$ is the list of nodes not yet expanded (frontier of search); $\hat{u}$ is the first element of $P$; $h$ is an evaluation of the criterion for a sequence from the current node to an end node; $g$ is the optimal value of the criterion from the origin node to the current node. $P$ is empty at the algorithm initialization.

The basic algorithm, inspired from the $A^*$ algorithm, is presented on Figure 7.

**begin**
  Search a first admissible sequence
  **if** a first admissible sequence has been found
    Initialize $BOUND$ to the found $J$
  **end**
  Put $n_1$ in P
  **while** P is not empty
    **for** each v in S($\hat{u}$)
    $T = \emptyset$
      **for** each possible action $a$
      Build sequence from $n_1$ to $v^a$
      Optimize $I$ choosing $t$ for each node of the sequence while meeting the constraints
      **if** there is a solution
      **then**
        Add $v^a$ in $T$
        $g = \hat{I}$
        Calculate $h$ from $v^a$ to an end node with a method $H_X$
          **if** $v^a \in W_e$ and $g < BOUND$
            $BOUND = g$
          **end**
        **end**
      **end**
      Prune the exploration tree
    **end**
    Put the elements of $T$ in P with a method $R_Y$
    Remove $\hat{u}$ from P
  **end**
**end**

Figure 7: Basic algorithm

## Proposed methods

Different methods are considered for the exploration strategy and for the pruning. Indeed, good exploration guidance and pruning provide good on-line replanning algorithm performances.

Cost evaluation methods, denoted $H_X$, calculate a cost evaluation $h$ of the plan from an unspecified node $n_i$ to an end node. For these methods, it is necessary to consider the criterion at a node $n_i$ of the sequence. The cost function $R_e(r_e)$ may be decomposed into a cost between the initial node and the current node $R_e(r_{\pi(i)})$ and a residual cost $\sigma_{\pi(i),e}$. The optimal value of $I$, denoted $g$, is an undervaluation of the value of:

$$R_e(r_{\pi(i)}) - \sum_{o \in E_o^i} R_o(t_{s(o)}, r_{s(o)}, t_{e(o)}, r_{e(o)}, t_{r(o)}, r_{r(o)})$$

for the plan that may be found on that branch of the tree.

Methods $H_1$ and $H_i$ aim at finding a lower bound of $J - g$. $H_1$ does not take into account the use of resources, while $H_i$

takes it into account until the current node $n_i$. It is possible to prove that:

$$J - g \geq \sigma_{\pi(i),e} - \sum_{o \in E_o \setminus E_o^i} R_o(t_{s(o)}, r_{s(o)}, t_{e(o)}, r_{e(o)}, t_{r(o)}, r_{r(o)})$$

As $E_o$ is unknown, $H_1$ and $H_i$ take into account the objectives of $P \setminus E_o^i$, knowing that this overvalues the number of treated objectives. For $H_1$, function $h$ is equal to:

$$- \sum_{o \in P \setminus E_o^i} \max_{t_{s(o)}, t_{e,o}, t_{r(o)}} R_o(t_{s(o)}, r_1, t_{e(o)}, r_1, t_{r(o)}, r_1)$$

As $\sigma_{\pi(i),e}$ is positive, method $H_1$ uses an upper bound of the sum of the rewards $R_o$ without taking into account the use of resources. The opposite of this bound is thus a lower bound of the value of the contribution to $J$ of the path from the current node $n_i$ to an end node. For $H_i$, function $h$ is equal to:

$$- \sum_{o \in P \setminus E_o^i} \max_{t_{s(o)}, t_{e(o)}, t_{r(o)}} R_o(t_{s(o)}, r_{\pi(i)}, t_{e(o)}, r_{\pi(i)}, t_{r(o)}, r_{\pi(i)})$$

Method $H_i$ uses an upper bound of the sum of the rewards taking into account the use of resources until the current node. The opposite of this bound is thus a more precise lower bound of the value of the contribution to $J$ of the path from $n_i$ to an end node.

Method $H_r$ is based on the solving of a relaxed problem: the problem is solved without optimizing the criterion $I$ at each node expansion and instants are chosen as if there were no constraint. When an end node is developed, instants are optimized for the found sequence. The difference between the calculated value of the criterion and the value $R_e(r_{\pi(i)})$ gives a value for $h$. Moreover, the number of developed nodes is fixed. Consequently, the value of $h$ is improved if other end nodes are developed with a better global criterion. Search will be better guided than for $H_1$ or $H_i$; however, the tree could be not pruned enough and search on all the possible sequences would be time-consuming. $H_r$ is neither a lower bound nor an upper bound of the criterion.

Method $H_W$ exploits the two-level hierarchy that describes the mission problem. The highest level describes the goal achievements level. The supernodes are the start planning node, one node for each objective and one node for the end of the mission. A superarc specifies a possible action between two supernodes. The superarcs have a weight that is a vector $A_r$ corresponding to the minimum use of resources on the arc and to the maximum reward obtained for the arc. A backward search is done on this supergraph. A heuristic value of the criterion is assigned to each supernode embodying a minimum use of resources and a maximum obtaining of rewards. This value is used during the search on the low level graph as $h$. It can be also used in order to prune the search tree. $H_W$ is a lower bound of the criterion.

Two pruning methods are used. The planner cannot apply traditional pruning methods as they suppose monotonic criterion function. The first pruning method is used if $h$ is a lower bound of the criterion from the current node to an end node. The rule applied by the algorithm is:

**if** $g + h > BOUND$ **then** prune node $v$.

The second one is used in other cases. The rule is:

**if** $(g + h) - \gamma|g + h| > BOUND$ **then** prune node $v$.

The choice of how to order the elements of $T$ in P is essential. If the arrangement is not efficient, the duration of the search may be high. Four arrangements are considered. $R_1$ and $R_2$ are ordered best-first searches guided by $g$ and $g+h$ respectively. They may be summed up in " sort $T$ in an increasing $g$ (or $g + h$) order and put $T$ on the top of P". $R_3$ and $R_4$ are $g$ (respectively $g + h$) best-first search strategies. They may be summed up in " put $T$ in P and sort P in an increasing $g$ (or $g + h$) order".

## Experiments

Some elementary tests have been performed and presented in (Chanthery, Barbier, & Farges 2004). They showed the capability of algorithms to make global planning in reasonable time for a small size problem. Here, the goal is to assess the efficiency of the algorithms in case of replanning for real missions. The performances of the algorithms are evaluated in limited time (15 minutes of maximum computation time), and measures of effectiveness are the values of the criterion for the first admissible solution and the best one found in the limited time and the times to obtain the first admissible path and the best path.

The mission is described by a take-off point $TOW$, which is the start point of the mission, a set of landing waypoints $LW$, which are the end points of the mission and a set of 9 objectives to try to fulfil. The replanning event happens after $O7$. There remain $O3$, $O4$, $O5$, $O6$, $O8$ and $O9$. Three tests of replanning are performed on the map of Figure 8. In the first test, $O3$ and $O6$ are cancelled. In the second test, $O3$ and $O6$ are cancelled and danger zones change: their radii increase from 20km to 40km and some danger zones moved. Finally, in the third test, an event of failure in the system induces a replanning. The vehicle has not enough fuel to finish its current plan that contains $O3$, $O4$, $O5$, $O6$, $O8$ and $O9$.

All the 16 possible algorithms are tested on these three scenarios on a SunBlade100 Processor. The value used for $\gamma$ is 0.001. The number of nodes developed during the first search and for $H_r$ method is 100. Tables 1, 2 and 3 present the results of the tests.

### General results analysis

With optimal speed, 20s of computation time correspond to about 1km on the map: this small position change is considered as acceptable. Indeed, it is unlikely that it changes drastically the planning problem and its solution. For all the tests, all algorithms that find a solution, except $H_W R_4$, find a first admissible solution in less than 20s. This first computation seems to be very efficient, as only method $H_r$ improves significantly that solution for scenarios 1 and 2.

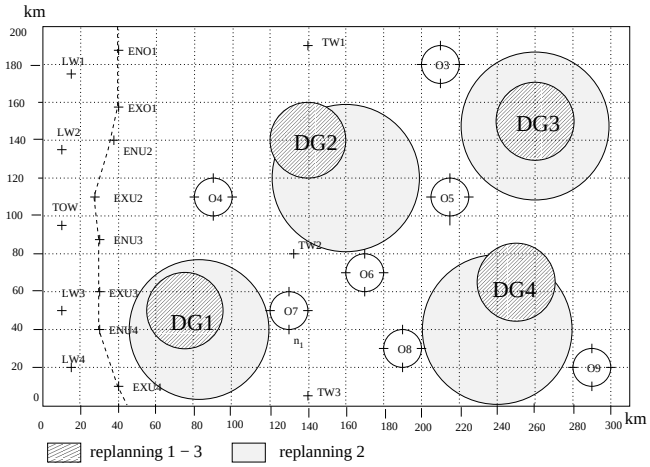



Figure 8: Tests map

| Algo | 1st adm sol | | best solution | |
|---|---|---|---|---|
| | J | time | J | time |
| $H_1R_1$ | -35898 | 9.5 | -35898 | 9.5 |
| $H_1R_2$ | -35898 | 10.3 | -35898 | 10.3 |
| $H_1R_3$ | -9449 | 1.8 | -9449 | 9.3 |
| $H_1R_4$ | -9449 | 2.8 | -9449 | 2.8 |
| $H_iR_1$ | -35898 | 9.9 | -35898 | 9.9 |
| $H_iR_2$ | -35898 | 9.5 | -35898 | 9.5 |
| $H_iR_3$ | -9449 | 1.8 | -9449 | 9.2 |
| $H_iR_4$ | -9449 | 2.7 | -9449 | 2.7 |
| $H_rR_1$ | -35898 | 9.5 | -36235 | 872.0 |
| $H_rR_2$ | -35898 | 9.6 | -36353 | 233.0 |
| $H_rR_3$ | -9449 | 1.9 | -9449 | 215.0 |
| $H_rR_4$ | -9449 | 2.7 | -36353 | 226.0 |
| $H_WR_1$ | -35898 | 9.7 | -35898 | 9.7 |
| $H_WR_2$ | 304453 | 7.6 | 304453 | 7.6 |
| $H_WR_3$ | -35898 | 9.9 | -35898 | 9.9 |
| $H_WR_4$ | 304453 | 7.6 | 304453 | 7.6 |

Table 2: Replanning 2: $O3$ and $O6$ are cancelled, danger $\nearrow$ - CPU time in seconds, J criterion value

| Algo | 1st adm sol | | best solution | |
|---|---|---|---|---|
| | J | time | J | time |
| $H_1R_1$ | -36160 | 12.9 | -36160 | 12.9 |
| $H_1R_2$ | -36161 | 12.9 | -36161 | 12.9 |
| $H_1R_3$ | -9449 | 1.9 | -9949 | 8.2 |
| $H_1R_4$ | -9449 | 2.8 | -9449 | 2.8 |
| $H_iR_1$ | -36161 | 13.0 | -36161 | 13.0 |
| $H_iR_2$ | -36161 | 13.2 | -36161 | 13.2 |
| $H_iR_3$ | -9449 | 1.9 | -9949 | 8.3 |
| $H_iR_4$ | -9449 | 2.8 | -9449 | 2.8 |
| $H_rR_1$ | -36161 | 12.9 | -36336 | 429.0 |
| $H_rR_2$ | -36161 | 12.7 | -36375 | 857.0 |
| $H_rR_3$ | -9449 | 1.9 | -9449 | 144.1 |
| $H_rR_4$ | -9449 | 2.8 | -36339 | 146.1 |
| $H_WR_1$ | -36161 | 12.9 | -36161 | 12.9 |
| $H_WR_2$ | -36176 | 8.8 | -36176 | 8.8 |
| $H_WR_3$ | -36161 | 13.0 | -36161 | 13.0 |
| $H_WR_4$ | -36175 | 8.8 | -36175 | 8.8 |

Table 1: Replanning 1: $O3$ and $O6$ are cancelled - CPU time in seconds, J criterion value

| Algo | 1st adm sol | | best solution | |
|---|---|---|---|---|
| | J | time | J | time |
| $H_1R_3$ | -9449 | 2.2 | -9449 | 10.1 |
| $H_1R_4$ | -9449 | 2.9 | -9449 | 296.1 |
| $H_iR_3$ | -9449 | 2.2 | -9449 | 10.2 |
| $H_iR_4$ | -9449 | 3.5 | -9449 | 297.4 |
| $H_rR_3$ | -9449 | 2.2 | -9449 | 425.0 |
| $H_rR_4$ | -9449 | 3.5 | -9449 | 3.5 |
| $H_WR_3$ | -9430 | 20.1 | -9430 | 20.1 |
| $H_WR_4$ | -9166 | 515.5 | -9166 | 515.5 |
| $H_xR_1$ | no solution | | | |
| $H_xR_2$ | no solution | | | |

Table 3: Replanning 3: limited fuel - CPU time in seconds, J criterion value

Methods $R_1$ and $R_2$ do not find any solution for the third test in the given computation time: ordered best-first search is not suited for too constrained problems.

As far as the quality of the best found solution is concerned, a variation of less than 1% compared with the best algorithm is considered as acceptable. $H_1$ and $H_i$ associated with $R_3$ and $R_4$ have the same behavior during the search and do not give results of good quality. $H_r R_4$ gives bad results for the first scenario and $H_w R_4$ for the second one.

### Choice of algorithms for future tests

The goal here is to find a set of algorithms that have good performances for all the tested situations.

First of all, methods $R_1$ and $R_2$ do not find any solution for the third test in the given computation time: they are thus rejected. Concerning the quality of the best found solution, methods $H_1$ and $H_i$ associated with $R_3$ and $R_4$ can be rejected. For the same reasons, $H_r R_4$ and $H_w R_4$ are also rejected.

The only two algorithms that seem to fit are $H_r R_4$ and $H_W R_3$. $H_r R_4$ has the advantage of improving the first solution, but $H_W R_3$ seems to be the best one, as far as these tests are concerned.

## Conclusions and Future Work

This paper presents a formalism for a class of real-world problems related to mission planning for autonomous vehicles. This formalism uses the notion of world abstraction with a two-level hierarchy. The highest level describes the goal achievement. This level is supplemented by a less abstract level that depends on the application and describes the exact motions of the vehicle in time and the use of resources. This work proposes a solution to a non-classical planning problem, where the number of objectives to fulfil is not *a priori* fixed. Moreover, the criterion is a non-monotonic function and planning has to deal with time and resources constraints. The classical pruning methods are therefore inefficient. The formalism is devoted to mission planning of autonomous vehicles. Its application in other application fields, for example project management, could be investigated.

Proposed algorithms use a variation of the standard heuristic search algorithm $A^*$. Those algorithms may be used for global planning at the beginning of the mission, but the main objective of the work is to use them on-line for replanning on order to react to events occurring during the mission. Different methods are considered for the exploration strategy and for the pruning and several methods of cost evaluation are proposed.

The formalism and the algorithms are applied to a military observation mission for an autonomous aerial system in a three dimensional, dynamic, uncertain and dangerous environment. Experiments are performed on this problem for several scenarios. The tests show that ordered best-first search is not adapted to too constrained replanning problems. Moreover, it seems that heuristics taking into account resources consumption combined with best-first search are acceptable. In general, these tests show how much on-line replanning is advantageous compared to classical treatments of unpredictable events, which recommend an emergency procedure for coming back in the safe area.

Future work will concern the tests of all the algorithms on a large set of missions and events in a real-time context. A fuzzy approach could be used for choosing the best algorithm for replanning, function of the environment, or function of the instant of occurrence and of the type of the event that is the cause of replanning. A balance should be found between the efficiency for pruning and the efficiency of the sorting methods. The solutions already tested could be compared to weighted $A^*$ with different weights for sorting and pruning. Algorithms not based on tree search may be proposed, for example using insertion or genetic approaches. The approach consisting into exploring the graph in breadth-first search and then applying the best partial solution when the instant for executing the first possible action is reached may be investigated.

## References


Barbier, M.; Lemaire, J.; and Toumelin, N. 2001. Procedures planner for an A.U.V. In *12th International Symposium on Unmanned Untethered Submersible Technology*.

Chanthery, E.; Barbier, M.; and Farges, J.-L. 2004. Mission planning for autonomous aerial vehicles. In *IAV2004 - 5th IFAC Symposium on Intelligent Autonomous Vehicles*.

Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2004. Autonomous management of an earth watching satellite. In *IAV2004 - 5th IFAC Symposium on Intelligent Autonomous Vehicles*.

Fikes, R., and Nilsson, N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*.

Frank, J., and Kurklu, E. 2003. Sofia's choice: Scheduling observations for an airborne observatory. In *13Th International Conference on Automated Planning & Scheduling*.

Frank, M., and Wolfe, P. 1956. *An Algorithm for quadratic programming*, volume 3. Naval Research Logistic Quaterly.

Galindo, C.; Fernández-Madrigal, J.; and González, J. 2004. Improving efficiency in mobile robot task planning through world abstraction. *IEEE Transaction on Robotics and Automation* 20(4):677–690.

Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning with numerical expressions in LPG. In *16th European Conference on Artificial Intelligence*.

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

Nilson, N. J. 1971. *Problem Solving Methods in Artificial Intelligence*. New-York: McGraw-Hill.

Schesvold, D.; Tang, J.; Ahmed, B. M.; Altenburg, K.; and Nygard, K. E. 2003. Pomdp planning for high level uav decisions: Search vs. strike. In *16th International Conference on Computer Applications in Industry and Engineering*.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In *14th International Conference on Automated Planning and Scheduling*.

Stentz, A. 2002. $CD^*$ : A real-time resolution optimal re-planner for globally constrained problems. In *AAAI-02*.

Teichteil, F., and Fabiani, P. 2004. An hybrid probabilistic model for autonomous exploration. In *RFIA 2004*.

van den Briel, M.; Nigenda, R.; Do, M.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In *AAAI*, 562–569.

Wah, B. W., and Chen, Y. 2004. Constrained partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence*.

# Responding to Uncertainty in UAV Surveillance through Desire Analysis

**David C. Han and K. Suzanne Barber**
The Laboratory for Intelligent Processes and Systems
The University of Texas at Austin
201 E. 24th St., ACE 5.402
Austin, Texas 78712
{dhan,barber}@lips.utexas.edu

**Abstract**

Uncertainty is inherent in multi-agent systems because agents do not know what actions other agents are going to perform. The Unmanned Aerial Vehicle (UAV) Surveillance domain is used to analyze two sources of uncertainty, (1) uncertainty of the costs of action due to target movement and (2) uncertainty of the rewards from goal achievement due to the actions of other agents. Decision-theoretic planning methods are presented to handle the first type of uncertainty. Modelling and analysis of the agent's desire structure is used to capture the effect of coordination on the second type of uncertainty.

## Introduction

The impact of uncertainty on the behavior of agents operating in multi-agent systems cannot be underestimated. Uncertainty is inherent in multi-agent systems because agents do not know what actions other agents are going to perform. Over time, the interests of an agent may change, changing the actions a rational agent should take in a given situation. As a simple example, consider two agents trying to complete the same task. If one agent knows the other agent will complete that task, it can apply its resources to other pursuits. In addition to interactions with other agents, mission updates from a commander will cause goals to be added, removed, or modified. Being autonomous entities, agents are given freedom to decide their own course of action for satisfying their goals and thusly must be equipped with facilities to respond to these changes.

Determining a course of action is a sequential decision problem, where the initial decision influences future decisions (i.e., the agent must consider not only the effects of its actions in the current state, but also the future consequences of any actions it takes). Further complicating the matter, the agent must consider the consequences of each action in relation to each of the goals the agent holds. Decision theory is the mathematical evaluation of risks, uncertainty, and benefits to calculate the value of alternative choices. Applied to agents, decision theory can form the basis for rational action selection in the face of uncertainty. An agent acts rationally if it performs actions that are in its "best interests" (Wooldridge 2000). Often defined in terms of their beliefs, desires, and intentions (Georgeff *et al.* 1999), the best interests of an agent correspond to the desires of the agent, modelled as goals the agent holds. Armed with decision theory, an agent can weigh the expected rewards to be gained from achieving each of its goals against the costs of actions to determine which goals are worth achieving, as well as the order in which to achieve those goals.

Markov decision processes (MDPs) (Feinberg & Schwartz 2002) are often used to represent and reason about sequential decision problems. MDPs inherently handle uncertainty by incorporating probabilities into the calculations of expected values for actions. Uncertainty is further captured by replacing reward values with "expected values" computed from probabilistic models of goal achievement.

However, there is a downside to using MDPs. Already suffering from the "curse of dimensionality," application of MDPs to domain problems containing multiple goals exacerbates the computational issues (by adding dimensions to the state representation). These additional dimensions, representing the achievement status of goals, do not reflect the behavior of the environment, but rather the internal state of the agent. The behavior of the environment (e.g., the domain physics) is conceptually different from the goals of an agent situated in that environment. It is due to limitations of the MDP reward structure that goal information must be represented as part of the state description.

Factoring can be used to reduce computation through abstraction of the state space. This research uses macro actions to factor the state space based on the desires (i.e., goals or tasks) of the agent into the *desire space* (Han & Barber 2004). The term desire space is used to maintain consistency with concepts from autonomous agents. The desire space is a representation of the possible achievement states and their interrelations due to the effect goal achievement order has on reward values accrued by the agent. Algorithms for modification of the desire space in response to goal addition, removal, and modification enable efficient recalculation of the desire space for action selection. For example, when information about the goals becomes known, the agent can efficiently update the desire-space model to reflect the most recent information and to maintain rationality of action.

The remainder of this paper is organized as follows. The next section describes the UAV surveillance domain which

motivates this research and to which this research is applied. This is followed by the a treatment of the foundations for decision-theoretic action selection and desire-space analysis for this research. Usage of the desire-space analysis is described for the purpose of modelling the effect of coordination on uncertainty related to the expected rewards. The last section summarizes and concludes the paper.

## UAV Surveillance Domain

The work presented in this paper is motivated by the domain of UAV (unmanned aerial vehicle) Surveillance and related navigation domains. The research in this paper has been implemented in a simulation of this domain for the purposes of experimentation and demonstration. Figure 1 shows the graphical user interface for the simulation. In the simulation, the environment has been abstracted into a Cartesian plane. Targets are placed at various points in this plane and the agents, controlling the UAVs, are tasked with visiting the various targets. An autonomous agent is assigned to each UAV to control its movement. At the most basic level, an agent has control over the heading and speed of a single UAV. A state in the state space is defined by the location, heading, and speed of the UAV in conjunction with the locations of the targets. Each target has an associated reward value, illustrated by the circles surrounding the targets (larger circles denotes larger rewards). Movement incurs a cost proportional to the distance traveled as an abstraction of resources, such as fuel, forcing the agents to consider trade off expected rewards against costs.

Uncertainty is introduced into the domain by the movement of the targets, forcing the UAVs to operate with possibly stale information regarding the location of the targets. If, upon arriving at the expected location of the target, the target is not present, the UAV must expend extra time and effort searching for the target. Three UAVs are shown in Figure 1. Lines extending out of the UAVs show the past and future planned movement.

Movement of the UAVs adds uncertainty to modelling the expected reward value received from visiting a target. Rewards are given only to the first UAV to visit each target so, unless an agent can predict the future actions of other agents, an agent is not guaranteed to receive any reward for its work. Calculating the exact expected cost incurred by an agent to reach a given target is rather complex due to the movement of the targets. Probabilistic encounter models could be used (E.g., cost for visiting a target can be estimated as a function of the distance between the UAV and the target).

As time progresses in the simulation, targets are added to reflect new objectives imposed on the UAVs by the mission commander. Upon being visited by a UAV, a target is removed from the system, representing the completion of that particular objective. As the UAVs are operating independently, they do not necessarily know when the targets will be removed from the system.

## Decision-Theoretic Planning

The UAV Surveillance domain, as described above is an over-subscription problem (Smith 2004), where there are a number of possible goals from which the agent must choose a subset to accomplish using its limited time and resources. Contrasting with other planning systems that attempt to find optimal solutions to this problem, this research is mainly concerned with the runtime effects that changing objectives have on the behavior of agents. Towards this end, abstraction is used to factor the decision space, decision-theoretic planning methods are used to reason at the goal level, and model modification (i.e., replanning) is examined.

Decision-theoretic planning, as described by Boutilier, Dean, and Hanks (Boutilier 1996) (Boutilier, Dean, & Hanks 1999), uses MDPs to perform this reasoning by allowing a range of reward values. As another benefit, MDPs naturally capture and handle the uncertainty inherent in the domain. Since the solution to an MDP consists of a policy describing the action to take in any given state, MDPs are suited for adaptation to continuous planning as well.

A Markov decision process $M$ is a representation of this action selection problem, consisting of four components: the state space, $S = \{s_1, s_2, ..., s_N\}$; actions the agent can execute,$A = \{a_1, a_2, ..., a_L\}$; a transition function describing the probability that executing each action $a$ in some state $s$ will result in some state $s'$, $T : S \times A \times S \mapsto [0, 1]$; and a reward function describing the value earned by the agent for reaching each state, $R : S \mapsto \mathbb{R}$. The product of an MDP planning algorithm is a policy $\pi : S \mapsto A$ describing what action the agent should execute for any state it may find itself in.

The cost function for the UAV Surveillance domain falls into the category of "cost-to-move" frameworks due to the structure the cost function imposes on the value calculations for the MDP. In cost-to-move problems, each action the agent executes incurs some cost $c < 0$ as part of the reward structure. This provides incentive for the agent to reach its goal states with the minimal amount of movement actions.

Even in the simplified UAV Surveillance domain, analysis of the state space directly is computationally intractable. Approximation and estimation methods are used to reduce computation required for decision-making in any reasonable time frame. Towards this end, this work addresses a restricted class of MDPs, using simple domains to explore complex goal related behaviors. Making the assumption that the set of goals assigned to an agent is much smaller than the total set of domain states, this research uses *macro actions* to abstract away the domain physics, reasoning about the desire space of the agent.

Macro actions are used to combine the primitive actions available to the agent, allowing the agent to reason at a higher level of abstraction (Sutton, Precup, & Singh 1999). Clever use of macro actions can improve computational efficiency for action selection. Macro actions are constructed for UAV Surveillance to move to each target location, enabling reasoning in the desire space. Through domain analysis, macro actions can be created manually to control the operation of the UAVs. The macros consist of the actions required to turn the UAV towards the specified target location and move until the destination is reached. If the target is not found at that location, then a simple search pattern

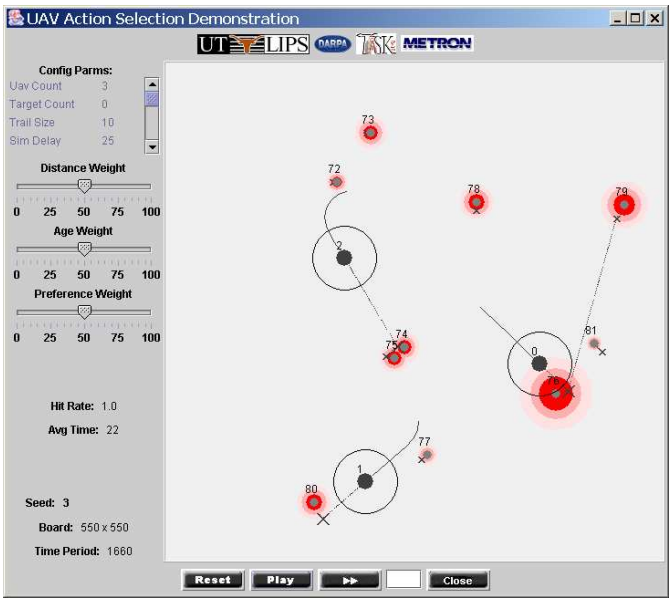



Figure 1: UAV demonstration of decision-theoretic action selection

is executed. Though not necessarily optimal, good domain analysis may yield macros that reasonably approximate the optimal behavior for each goal.

## Macro Actions and the Desire Space

The use of the basic reward structure for MDP models is limiting in that if the agent has multiple goals to achieve, those goals must be represented as part of the state definition. For example, consider if the domain states are defined as a product of state variables, $S_{domain} = V_1 \times V_2 \times ... \times V_L$. If an agent desires to sequentially visit multiple states in the domain, the actions that the agent selects will be different depending on which of the goal states the agent has already visited. Desire states of the agent can be defined as a product of the goal variables (boolean values indicating whether each goal has been achieved), $S_{desire} = G_1 \times G_2 \times ... \times G_K$. The states represented in MDP $M$ must be able to differentiate between the same domain states when the agent has different desire states, hence $S = V_1 \times V_2 \times ... \times V_L \times G_1 \times G_2 \times ... \times G_K$. In essence, the additional propositions from $S_{desire}$ are used to prevent attempted repeat collection of the same reward and are necessary to accurately model the domain.

Computation for solving an MDP is dependent upon its size. Factoring can been used to reduce computation through abstracting the MDP into higher level states and actions. This research makes use of the concept of macro actions, specifically, the *option* model developed by Sutton, Precup, and Singh (Sutton, Precup, & Singh 1999). Macro actions generalize actions into courses of action, combining primitive actions to reach some objective, such as moving from landmark to landmark. To construct the desire space, the target (goal) locations are used as the landmarks for generation of macro actions (Han & Barber 2004).

For example, take the domain illustrated in Figure 2 as a further simplification of the UAV Surveillance domain. Locations labelled **1** through **3** represent the goals a robot desires to visit. Unlike primitive actions, execution of a macro action will have variable cost depending on the distance from the state in which the macro action was initiated to the goal location. Assuming uniform cost per move, the cost for execution of a macro is equal to the cost per move times the expected number of moves from the current state to the termination state (Equation 1). One other benefit of using a macro action is that the uncertainty of action in the domain is encapsulated in the expected cost function.

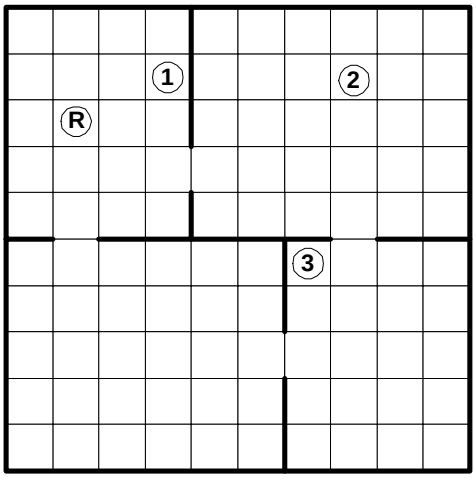



Figure 2: Multiple goals in a navigation task

$$C(macro_1, s) = cE(\text{\# of moves from } s \text{ to termination}) \quad (1)$$

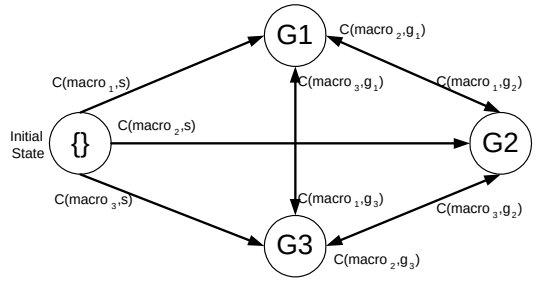



Figure 3: Weighted graph relating costs to travel among the goal locations

The expected costs of macro actions are used to form a weighted graph among all the subgoals as shown in Figure 3. This represents the reduced state space after the factoring using the macro actions. Many algorithms exist for finding the shortest path visiting all nodes in a graph (Gutin & Punnen 2002). The application of travelling salesman algorithms determines the order for a compound macro action which describes a sequential ordering of macro actions. While this is useful for goals that exist in an 'AND' (i.e., reward is not given unless all subgoals have been achieved) relationship, this does not reward partial achievement. If goals are independent, or related in a more complex manner, the agent should be able to reason about the expected rewards for handling a subset of the targets.

## Desire States and Goal Analysis

An important characteristic of an autonomous agent is the ability to decide which goals to pursue. Towards this end, the agent's desires may be combined in an 'OR' fashion, where the agent may receive rewards for goals independent of other goals . In this case, the agent must consider not only the order in which to achieve goals, but whether to try to achieve each particular goal at all - the cost to achieve a goal may outweigh the reward. Additionally, since execution of actions will change the agent's distance and thus the cost to reach the respective goals, pursuing one goal may make it more or less profitable (even unprofitable) to pursue other goals.

By creating macro actions to achieve each individual goal, the entire set of state variables (and their uncertainty) can be abstracted away. Instead, reasoning can be performed purely in terms of desire states, referred to in this paper as the *desire space*. Figure 4 shows the desire space for the example navigation domain shown in Figure 2. Each state is labelled with the set of goal variables denoting which goals have yet to be achieved in that state. Initially, the agent is in the state marked by the full set of goals and the current location. Application of each macro leads the agent to the desire state where the appropriate goal is marked as achieved, leading up to the state with all goals being achieved. Unfortunately, the domain space cannot be completely factored out because the expected cost function for the macro actions is dependent upon domain state. Luckily, if an agent executes actions according to this decision-making mechanism, the only relevant states are the current state and the termination states of the macro actions, resulting in a much reduced space to search.

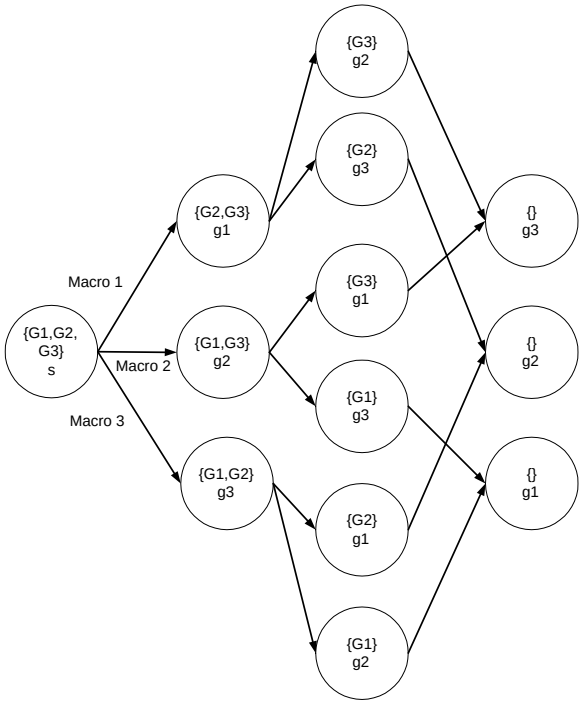



Figure 4: Desire space for the three goal navigation domain

The motivations for reasoning in the desire space include: (1) the desire space is smaller than the complete state space (the desire space grows in the number of tasks, not the number of state variables), and (2) the structure of the desire space can be exploited algorithmically during computation. The model for reasoning about the desire space is defined as follows. Given the domain space of the problem $S_{domain}$, some subset of those states are marked as goals, $G \subseteq S_{domain} = \{g_1, g_2, ...g_K\}$. Each goal represents one target in the UAV domain. Although the locations are uncertain due to the movement of the targets, macro actions allow the goals to be modelled in a semi-deterministic manner. The terminal states are represented as a probability distribution over the domain states. However, due to the nature of macro actions, the probability is concentrated on the goal state. It is possible for a macro to have termination states that represent failure of that macro to achieve its goal but, for simplicity of explanation, this paper expects the macro actions to always terminate in its goal state without fail. In this case, the last known location of the target is used as the goal state and the uncertainty is captured in the cost estimation function for the macros, where the number of moves necessary to visit a target is probabilistically modelled.

The states of the desire space are built from the goal variables (achievement state) and the agent's location in the domain space. Each macro action is constructed to move the agent to a given goal state. The desire states are denoted by a tuple $\langle G_{unach}, s \rangle$. The first element of the tuple, $G_{unach}$ is the set of goals that have not been achieved. The second element of the tuple is the location of the agent in $S_{domain}$. The agent can only be located at the initial location $s_{initial}$, or as a result of executing a macro action, in an accomplished goal location $g_i$, hence, $S_{desire} = \{\langle G, s_{initial} \rangle, \langle G_{unach}, g_i \rangle$ s.t. $G_{unach} \subseteq G$ and $g_i \in Goals/G_{unach}\}$. The action set $A_{desire} = \{macro_1, macro_2, \ldots, macro_K\}$ is the set of macro actions, one for achieving each goal the agent holds. Finally, the reward function, $R : Goals \mapsto \mathbb{R}$, assigns a separate reward value to each goal. An action level cost function $c_{action}$ is required to estimate the costs incurred by executing the macro action. This cost is related to the distance the agent must travel from a given domain state to the termination state of the macro.

Since the reward function is assigned slightly differently from that used in a standard MDP, the evaluation of states and actions is changed to match. Global termination states are those desire states in which there are no profitable macro actions. States in which all goals have been achieved are global termination states since all rewards have already been collected. The global termination states (where all goals have been achieved) are assigned a value of 0, indicating that no further action will yield any future rewards. Under the Markovian assumption, state transitions are not dependent upon history. After an agent has executed some action, the action that was executed becomes irrelevant and future decisions are dependent only upon the resulting state. As a side effect, an agent may perform an immediately unprofitable action in expectation of higher future rewards, but will never take an immediately unprofitable action based on highly profitable past action. Consequently, rational action requires that the tail (of any length) of an agent's plan will always be profitable. The expected value of desire states is defined in equations 2 and 3.

The value of a state is simply the sum of the cost of executing the macro from that state (a negative number), the reward for achieving the immediate goal through macro execution, and any expected value for being in the resulting state, due to expected future goal achievement. Note that if no action is profitable (i.e., the cost of each action outweighs or equals its benefits), then the state is also a global termination state and is given a value of 0.

The specific structure of the graph offers many exploitable characteristics. Since the domain does not allow goals to become unachieved, loops cannot exist in the graph, forming a tree structure. This enables calculation of the expected values to proceed through simple accumulation of the values from a single graph traversal.

## Model Modification for Dynamic Objectives

In UAV Surveillance, targets are added to the system as the mission is updated. This reflects shifts in priorities by the mission commander which are then passed down to the UAVs. Additionally, with more than one UAV operating in the system, targets may be removed through the action of the other agents (i.e., when another agent visits a target and receives the reward). These changes must be reflected in the decision model used by the agents to maintain rational operation. The probability of receiving the reward value for goals as specified change over time as information from the mission commander and from other agents is used to reduce the uncertainty surrounding each of the goals. The following sections describe algorithms for adding and removing goals to the desire-space model during runtime.

### Goal Removal

Goal removal allows the agent to reduce the size of the desire space that it models. There are two cases for goal removal: (1) the goal has already been achieved or (2) the goal has not already been achieved. The second case may occur if the goal is being abandoned, mooted by the commander, or contracted to another agent. Both cases are simple due to the structure of the desire space.

The first case is trivial due to the structure of the desire space. The agent needs only treat the current state as the new root of the model with no recalculation necessary. All desire states that are not reachable from the current desire state can be pruned from the model (e.g., all those desire states the goal being removed contributes value to). In fact, the goal variable itself can be removed from the representation. Since the value assigned to that goal variable will be equivalent for all remaining states, it can be safely factored out of the desire state representation without affecting any of the remaining desire state values.

**Algorithm 1** REMOVEGOAL($d$,$g$)

$location = d.location$
$d =$CHILD($d, g$)
$d.location = location$
UPDATE($V(d)$)

When the goal being removed has not already been achieved (i.e., it is being adopted by another agent or abandoned), recalculation is necessary to remove the value of the goal from the action-selection reasoning. Due to the structure of the desire space (Figure 4), the value of any given node is dependent only on the unachieved goals and state of the agent at that node. Computation is saved by caching the values of each node. Algorithm 1 describes the removal of goal $g$. The function CHILD($d, g$) selects and returns the desire state that results from executing the macro to achieve $g$ in the desire state $d$. The agent transitions in the desire space as if it had achieved goal $g$. The resulting state in the desire space is then updated with the agent's current location in the state space. Finally, the value of the current new state is recalculated based on the new location. The values of the children states had previously been calculated, but due to the new location, the costs to reach the children have changed. This may cause a new macro to be selected as the most profitable when calculating the new $V(d)$.

$$V(\langle\{\}, s\rangle) = 0 \tag{2}$$

$$V(\langle G_{unach}, s\rangle) = \max\left(0, \max_{macro_i \in A_{desire}} \begin{pmatrix} c_{action}(macro_i, s) \\ +R(g_i) \\ +V(\langle G_{unach} - g_i, g_i\rangle) \end{pmatrix}\right) \tag{3}$$

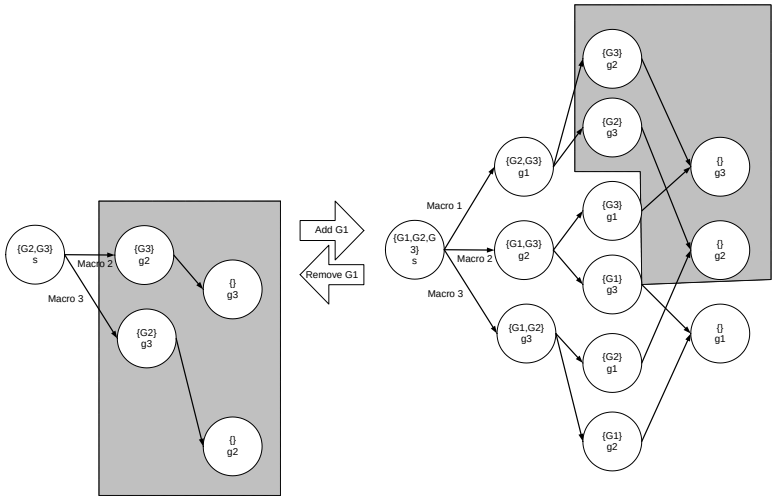



Figure 5: Modification of desire space for addition or removal of a goal

## Goal Addition

Algorithm 2 describes a process for adding goal $g$ to desire state $d$. For desire state $d$ in the model, a new macro action is added for achieving goal $g$ and the resulting desire state $d'$ is created. The children of $d$ are added to $d'$. After the addition of the children, the value of $d'$ can be calculated, selecting the best macro to execute in that desire state. The new goal $g$ are then added to each of the children of $d$, constructing the model while executing depth-first traversal of the tree. Finally, the value of $s$ is updated, possibly changing the best macro to execute.

**Algorithm 2** AddGoal($d$,$g$)

$d' =$ new STATE($\langle d.G_{unach}, g\rangle$)
$d.G_{unach} = d.G_{unach} + g$
**for all** $i \in d.children$ **do**
  ADDCHILD($d', i$)
**end for**
UPDATE($V(d')$)
**for all** $i \in d.children$ **do**
  ADDGOAL($i$,$g$)
**end for**
$d.children = d.children + d'$
UPDATE($V(d)$)

Model modification saves computational cost compared to building a new model by reusing calculations for subpaths that do not contain the new task. Figure 5 shows the result of adding $g_1$ to a model that already includes $g_2$ and $g_3$. Desire states marked in gray are replicated from the original model into the resulting model through ADDCHILD in the algorithm described above.

Additionally, since values are accumulated from the end of the path back towards the head of the path, some desire state nodes are shown with multiple incoming edges. The value for these nodes needs only be calculated a single time, cached, then reused for each of the incoming edges. Replication saves the computational cost of recalculating the values for states which will have equivalent values to preexisting states.

Algorithm 2 is essentially a depth-first search, but was included to illustrate how new nodes are added into the model during the search process. Heuristic usage can modify the presented algorithm to a best-first search to further reduce computational costs, though the complexity level is not changed.

## Goal Modification

The rewards associated with goals may change. This may be due to the passage of time or acquisition of new information. States in which the goal has been achieved are not affected by any change in the value of that goal. Only those states leading up to achievement of that goal are affected. Similar to the addition of goals, desire state values can be updated by a single traversal of the graph. By intelligently caching the value calculation results large sections of the desire space are not touched.

The overall objective when handling dynamic goals is to reuse the calculations that stay static across changes. In each of the removal, addition, or modification cases, the desire space is divided into sections by the appropriate macro ac-

tion. On the originating side of the macro action, desire states require recalculation. On the resulting side of the macro action, previously calculated values can be reused.

## Controlling Uncertainty through Coordination

While the uncertainty of the UAV Surveillance domain is captured by decision-theoretic planning, the uncertainty from the action of other agents is not. Using the methods of goal addition, removal, and modification, coordination among the agents can be used to reduce uncertainty. Based on the locations of the UAVs and any messages passed between the agents, an agent can adjust the expected reward values in its desire space to reflect the probable actions of other agents.

Different levels of coordination may be employed, depending on the requirements on resource consumption and solution quality. The purpose of coordination is to reduce uncertainty about the expected rewards for visiting each target. Four types of coordination are examined and their ability to reduce uncertainty evaluated: (1) no coordination, (2) location-based inference, (3) communicated inference, and (4) explicit partitioning.

With no coordination, the agents operate without any knowledge of the other agents in the system. This option requires no additional computational resources or communication on behalf of the agents. Since the agents have no awareness of the other agents, they tend to operate redundantly, often attempting to visit the same target. This situation reflects the most uncertainty.

Location-based inference and communicated inference both produce an implicit partitioning of the goals, reducing the overlap in work performed by the agents when compared to no coordination. Location-based inference uses only information about the physical location of the UAVs and the targets. Targets that are closer to other agents have their expected rewards reduced due to the increased probability that the other agents will visit those targets first. Communicated inference is similar to location-based inference, but the agents calculate which are their preferred targets and communicate those preferences to the other agents. The benefit of this over location-based inference is that the agents can take their paths (i.e., their future locations) into account when calculating their preferences instead of just their present location. In these two situations, the agents suffer somewhat less uncertainty than in the case of no coordination.

With explicit partitioning, the agents negotiate an allocation of the goals to respective agents effectively reducing the overlap to zero. One drawback of using explicit partitioning is an increase in both communications the additional computational resources needed to calculate and negotiate the partition. Also, this method can result in commitments far into the future, reducing performance of the agents restricting the ability to adapt to changing conditions quickly. This situation represents the least uncertainty.

Figures 6 and 7 compare the four coordination mechanisms described above. In each case, three agents are used to cover a battlefield. Targets are added to random locations

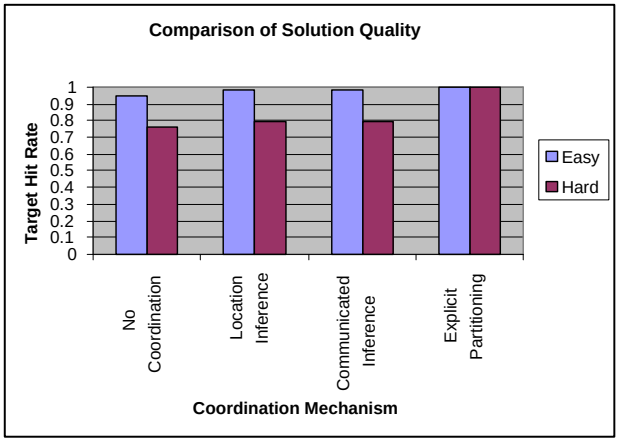



Figure 6: Comparison of the quality of solution as a percentage of the rewards received by the multi-agent system

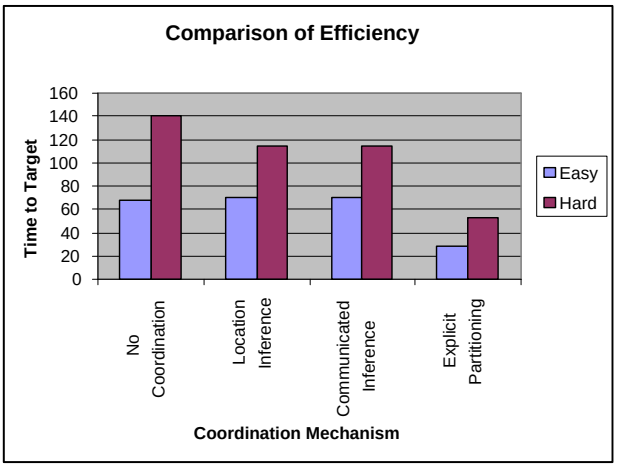



Figure 7: Comparison of the efficiency of solution as an average of the costs incurred per goal

on the battlefield at regular intervals. Difficulty of coverage was set for the agents by the speed at which targets are added. Targets have a given lifetime after which, if they have not been visited by an UAV, they are removed by the mission commander. If this occurs, it is counted as a missed target. Figure 6 shows the effect of the coordination mechanisms on the ability for the agents to spread out across the battlefield. The results show that explicit partitioning is the best, while the implicitly partitioning of location inference and communicated inference are slightly better than no coordination. Figure 7 shows the efficiency of the agents at retrieving their rewards, measuring the distance travelled on average to visit each target since cost is dependent upon distance. Increasing the amount of coordination reduces the distance travelled, meaning there was less overlap in the actions of the agents due to less uncertainty about the actions of other agents.

## Conclusions

Multi-agent systems operate in dynamic environments. Dynamism may come from many sources, resulting in uncertainty in the system. Using UAV Surveillance as an example domain, the uncertainty dealt with in this paper originate from two sources, (1) the movement of the targets results in uncertainty on the cost required to service that target and (2) the actions of the other agents results in uncertainty about the rewards expected for each goal.

Agents must respond to the uncertainties they perceive and try to act as best they can. This paper described approaches to handle both the uncertainty in costs and the uncertainty in rewards. Decision-theoretic planning is used to handle the uncertainty from the domain, while desire-space modelling techniques provide a means for the agents to model reductions in uncertainty of rewards through coordination and improve their action selection quality to match.

## Acknowledgements

This research was funded in part by the Defense Advanced Research Projects Agency and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0588. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed on implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

The research reported in this document was performed in connection with Contract number DAAD13-02-C-0079 with the U.S. Edgewood Biological Chemical Command.

## References

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Boutilier, C. 1996. Planning, learning and coordination in multiagent decision processes. In *Theoretical Aspects of Rationality and Knowledge*, 195–201.

Feinberg, E. A., and Schwartz, A., eds. 2002. *Handbook of Markov Decision Processes*. Boston, MA: Kluwer.

Georgeff, M. P.; Pell, B.; Pollack, M. E.; Tambe, M.; and Wooldridge, M. 1999. The belief-desire-intention model of agency. In *ATAL '98: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, 1–10. London, UK: Springer-Verlag.

Gutin, G., and Punnen, A. P., eds. 2002. *The Traveling Salesman Problem and Its variations*. Dordrecht, The Netherlands: Kluwer.

Han, D., and Barber, K. S. 2004. Desire-space analysis and action selection for multiple, dynamic goals. In *CLIMA V, Computational Logic in Multi-agent Systems*, 182–195.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *ICAPS*, 393–401. AAAI.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2):181–211.

Wooldridge, M. 2000. *Reasoning about Rational Agents*. Cambridge, Massachusetts: The MIT Press.

# An AO* Algorithm for Planning with Continuous Resources

**Emmanuel Benazera**[*]
**Ronen Brafman**[†]**, Nicolas Meuleau**[†]
NASA Ames Research Center
Mail Stop 269-3
Moffet Field, CA 94035-1000
{ebenazer, brafman, nmeuleau}
@email.arc.nasa.gov

**Mausam**
Dept. of Computer Science
and Engineering
University of Washington
Seattle, WA 981952350
mausam@cs.washington.edu

**Eric A. Hansen**
Dept. of Computer Science
and Engineering
Mississippi State University
Mississippi State, MS 39762
hansen@cse.msstate.edu

**Abstract**

We consider the problem of optimal planning in stochastic domains with resource constraints, where resources are continuous and the choice of action at each step may depend on the current resource level. Our principal contribution is the HAO* algorithm, a generalization of the AO* algorithm that performs search in a hybrid state space that is modeled using both discrete and continuous state variables. The search algorithm leverages knowledge of the starting state to focus computational effort on the relevant parts of the state space. We claim that this approach is especially effective when resource limitations contribute to reachability constraints. Experimental results show its effectiveness in the domain that motivates our research – automated planning for planetary exploration rovers.

## Introduction

Control of planetary exploration rovers presents several important challenges for research in automated planning. Because of difficulties inherent in communicating with devices on other planets, remote rovers must operate autonomously over substantial periods of time (Bresina *et al.* 2002). The planetary surfaces on which they operate are very uncertain environments: there is a great deal of uncertainty about the duration, energy consumption, and outcome of a rover's actions. Currently, instructions sent to planetary rovers are in the form of a simple plan for attaining a single goal (e.g., photographing some interesting rock). The rover attempts to carry this out, and, when done, remains idle. If it fails early on, it makes no attempt to recover and possibly achieve an alternative goal. This may have a serious impact on missions. For example, it has been estimated that the 1997 Mars Pathfinder rover spent between 40% and 75% of its time doing nothing because plans did not execute as expected. The current MER rovers (*aka* Spirit and Opportunity) require an average of 3 days to visit a single rock, but in future missions, multiple rock visits in a single communication cycle will be possible (Pedersen *et al.* 2005). As a result, it is expected that space scientists will request a large number of potential tasks for future rovers to perform, more than may be feasible, presenting an oversubscribed planning problem.

Working in this application domain, our goal is to provide a planning algorithm that can generate reliable contingent plans that respond to different events and action outcomes. Such plans must optimize the expected value of the experiments conducted by the rover, while being aware of its time, energy, and memory constraints. In particular, we must pay attention to the fact that given any initial state, there are multiple locations the rover could reach, and many experiments the rover could conduct, *most combinations of which* are infeasible due to resource constraints. To address this problem we need a faithful model of the rover's domain, and an algorithm that can generate optimal or near-optimal plans for such domains. General features of our problem include: (1) a concrete starting state; (2) continuous resources (including time) with stochastic consumption; (3) uncertain action effects; (4) several possible one-time-rewards, only a subset of which are achievable in a single run. This type of problem is of general interest, and includes a large class of (stochastic) logistics problems, among others.

Past work has dealt with some features of this problem. Related work on MDPs with resource constraints includes the model of constrained MDPs developed in the OR community (Altman 1999). A constrained MDP is solved by a linear program that includes constraints on resource consumption, and finds the best feasible policy, given an initial state and resource allocation. A drawback of the constrained MDP model is that it does not include resources in the state space, and thus, a policy cannot be conditioned on resource availability. Moreover, it does not model stochastic resource consumption. In the area of decision-theoretic planning, several techniques have been proposed to handle uncertain continuous variables (e.g. (Feng *et al.* 2004; Younes and Simmons 2004; Guestrin *et al.* 2004)). Smith 2004 and van den Briel et *al.* 2004 consider the problem of oversubscription planning, i.e., planning with a large set of goals which is not entirely achievable. They provide techniques for selecting a subset of goals for which to plan, but they deal only with deterministic domains. Finally, Meuleau *et al.* 2004 present preliminary experiments towards scaling up decision-theoretic approaches to planetary rover problems.

Our contribution in this paper is an implemented algorithm, Hybrid AO* (HAO*), that handles all of these prob-

[*] Research Institute for Advanced Computer Science.
[†] QSS Group Inc.

lems together: oversubscription planning, uncertainty, and limited continuous resources. Of these, the most essential features of our algorithm are its ability to handle hybrid state-spaces and to utilize the fact that many states are unreachable due to resource constraints.

In our approach, resources are included in the state description. This allows decisions to be made based on resource availability, and it allows a stochastic resource consumption model (as opposed to constrained MDPs). Although this increases the size of the state space, we assume that the value functions may be represented compactly. We use the work of Feng et *al*. (2004) on piecewise constant and linear approximations of dynamic programming (DP) in our implementation. However, standard DP does not exploit the fact that the reachable state space is much smaller than the complete state space, especially in the presence of resource constraints. Our contribution is to show how to use the forward heuristic search algorithm called AO* (Pearl 1984; Hansen and Zilberstein 2001) to solve MDPs with resource constraints and continuous resource variables. Unlike DP, forward search keeps track of the trajectory from the start state to each reachable state, and thus it can check whether the trajectory is feasible or violates a resource constraint. This allows heuristic search to prune infeasible trajectories and can dramatically reduce the number of states that must be considered to find an optimal policy. This is particularly important in our domain where the discrete state space is huge (exponential in the number of goals), yet the portion reachable from any initial state is relatively small because of the resource constraints. It is well-known that heuristic search can be more efficient than DP because it leverages a search heuristic and reachability constraints to focus computation on the relevant parts of the state space. We show that for problems with resource constraints, this advantage can be even greater than usual because resource constraints further limit reachability.

The paper is structured as follows: In Section 2 we describe the basic action and goal model. In Section 3 we explain our planning algorithm, HAO*. Initial experimental results are described in Section 4, and we conclude in Section 5.

## Problem Definition and Solution Approach

### Problem Formulation

We consider a Markov decision process (MDP) with both continuous and discrete state variables (also called a *hybrid MDP* (Guestrin *et al.* 2004) or *Generalized State MDP* (Younes and Simmons 2004)). Each state corresponds to an assignment to a set of state variables. These variables may be discrete or continuous. Continuous variables typically represent resources, where one possible type of resource is time. Discrete variables model other aspects of the state, including (in our application) the set of goals achieved so far by the rover. (Keeping track of already-achieved goals ensures a Markovian reward structure, since we reward achievement of a goal only if it was not achieved in the past.) Although our models typically contain multiple discrete variables, this plays no role in the description of our algorithm, and so, for notational convenience, we model the discrete component as a single variable $n$.

A *Markov state* $s \in S$ is a pair $(n, \mathbf{x})$ where $n \in N$ is the discrete variable, and $\mathbf{x} = (x_i)$ is a vector of continuous variables. The domain of each $x_i$ is an interval $X_i$ of the real line, and $\mathbf{X} = \bigotimes_i X_i$ is the hypercube over which the continuous variables are defined. We assume an explicit *initial state*, denoted $(n_0, \mathbf{x}_0)$, and one or more absorbing *terminal states*. One terminal state corresponds to the situation in which all goals have been achieved. Others model situations in which resources have been exhausted or an action has resulted in some error condition that requires executing a safe sequence by the rover and terminating plan execution.

*Actions* can have executability constraints. For example, an action cannot be executed in a state that does not have its minimum resource requirements. $A_n(\mathbf{x})$ denotes the set of actions executable in state $(n, \mathbf{x})$.

*State transition probabilities* are given by the function $\Pr(s' \mid s, a)$, where $s = (n, \mathbf{x})$ denotes the state before action $a$ and $s' = (n', \mathbf{x}')$ denotes the state after action $a$, also called the arrival state. Following (Feng *et al.* 2004), the probabilities are decomposed into:

- the discrete marginals $\Pr(n'|n, \mathbf{x}, a)$. For all $(n, \mathbf{x}, a)$, $\sum_{n' \in N} \Pr(n'|n, \mathbf{x}, a) = 1$;
- the continuous conditionals $\Pr(\mathbf{x}'|n, \mathbf{x}, a, n')$. For all $(n, \mathbf{x}, a, n')$, $\int_{x' \in \mathbf{X}} \Pr(\mathbf{x}'|n, \mathbf{x}, a, n') d\mathbf{x}' = 1$.

Any transition that results in negative value for some continuous variable is viewed as a transition into a terminal state.

The *reward* of a transition is a function of the arrival state only. More complex dependencies are possible, but this is sufficient for our goal-based domain models. We let $R_n(\mathbf{x}) \geq 0$ denote the *reward* associated with a transition to state $(n, \mathbf{x})$.

In our application domain, continuous variables model non-replenishable resources. This translates into the general assumption that the value of the continuous variables is non-increasing. Moreover, we assume that each action has some minimum positive consumption of at least one resource. We do not utilize this assumption directly. However, it has two implications upon which the correctness of our approach depends: (1) the values of the continuous variables are a-priori bounded, and (2) the number of possible steps in any execution of a plan is bounded, which we refer to by saying the problem has a *bounded horizon*. Note that the actual number of steps until termination can vary depending on actual resource consumption.

Given an initial state $(n_0, \mathbf{x_0})$, the objective is to find a policy that maximizes expected cumulative reward.[1] In our application, this is equal to the sum of the rewards for the goals achieved before running out of a resource. Note that there is no direct incentive to save resources: an optimal solution would save resources only if this allows achieving more goals. Therefore, we stay in a standard decision-theoretic framework. This problem is solved by solving Bellman's optimality equation, which takes the following

[1] Our algorithm can easily be extended to deal with an uncertain starting state, as long as its probability distribution is known.

form:

$$V_n^0(\mathbf{x}) = 0 \ ,$$

$$V_n^{t+1}(\mathbf{x}) = \max_{a \in A_n(\mathbf{x})} \left[ \sum_{n' \in N} \Pr(n' \mid, n, \mathbf{x}, a) \int_{\mathbf{x}'} \Pr(\mathbf{x}' \mid n, \mathbf{x}, a, n') \left( R_{n'}(\mathbf{x}') + V_{n'}^t(\mathbf{x}') \right) d\mathbf{x}' \right] \ . \quad (1)$$

Note that the index $t$ represents the iteration or *time-step* of DP, and does not necessarily correspond to time in the planning problem. The duration of actions is one of the biggest sources of uncertainty in our rover problems, and we typically model time as one of the continuous resources $x_i$.

### Solution Approach

Feng *et al.* describe a dynamic programming (DP) algorithm that solves this Bellman optimality equation. In particular, they show that the continuous integral over $\mathbf{x}'$ can be computed exactly, as long as the transition function satisfies certain conditions. This algorithm is rather involved, so we will treat it as a black-box in our algorithm. In fact, it can be replaced by any other method for carrying out this computation. This also simplifies the description of our algorithm in the next section and allows us to focus on our contribution. We do explain the ideas and the assumptions behind the algorithm of Feng *et al.* in Section 3.

The difficulty we address in this paper is the potentially *huge* size of the state space, which makes DP infeasible. One reason for this size is the existence of continuous variables. But even if we only consider the discrete component of the state space, the size of the state space is exponential in the number of propositional variables comprising the discrete component. To address this issue, we use forward heuristic search in the form of a novel variant of the AO* algorithm. Recall that AO* is an algorithm for searching AND/OR graphs (Pearl 1984; Hansen and Zilberstein 2001). Such graphs arise in problems where there are choices (the OR components), and each choice can have multiple consequences (the AND component), as is the case in planning under uncertainty. AO* can be very effective in solving such planning problems when there is a large state space. One reason for this is that AO* only considers states that are reachable from an initial state. Another reason is that given an informative heuristic function, AO* focuses on states that are reachable in the course of executing a good plan. As a result, AO* often finds an optimal plan by exploring a small fraction of the entire state space.

The challenge we face in applying AO* to this problem is the challenge of performing state-space search in a continuous state space. Our solution is to search in an *aggregate state space* that is represented by a search graph in which there is a node for each distinct value of the discrete component of the state. In other words, each node of our search graph represents a region of the continuous state space in which the discrete value is the same. In this approach, different actions may be optimal for different Markov states in the aggregate state associated with a search node, especially since the best action is likely to depend on how much energy or time is remaining. To address this problem and still find an optimal solution, we associate a value estimate with each of the Markov states in an aggregate. That is, we attach to each search node a value function (function of the continuous variables) instead of the simple scalar value used by standard AO*. Following the approach of (Feng *et al.* 2004), this value function can be represented and computed efficiently due to the continuous nature of these states and the simplifying assumptions made about the transition functions. Using these value estimates, we can associate different actions with different Markov states within the aggregate state corresponding to a search node.

In order to select which node on the fringe of the search graph to expand, we also need to associate a scalar value with each search node. Thus, we maintain for a search node both a heuristic estimate of the value function (which is used to make action selections), and a heuristic estimate of the priority which is used to decide which search node to expand next. Details are given in the following section.

We note that LAO*, a generalization of AO*, allows for policies that contain "loops" in order to specify behavior over an infinite horizon (Hansen and Zilberstein 2001). We could use similar ideas to extend LAO* to our setting. However, we need not consider loops for two reasons: (1) our problems have a bounded horizon; (2) an optimal policy will not contain any intentional loop because returning to the same discrete state with fewer resources cannot buy us anything. Our current implementation assumes any loop is intentional and discards actions that create such a loop.

## Hybrid AO*

A simple way of understanding HAO* is as an AO* variant where states with identical discrete component are expanded in unison. HAO* works with two graphs:

- The *explicit graph* describes all the states that have been generated so far and the AND/OR edges that connect them. The nodes of the explicit graph are stored in two lists: OPEN and CLOSED.
- The *greedy policy* (or partial solution) graph, denoted GREEDY in the algorithms, is a sub-graph of the explicit graph describing the current optimal policy.

In standard AO*, a single action will be associated with each node in the greedy graph. However, as described before, multiple actions can be associated with each node, because different actions may be optimal for different Markov states represented by an aggregate state.

### Data Structures

The main data structure represents a search node $n$. It contains:

- The value of the discrete state. In our application these are the discrete state variables and set of goals achieved.
- Pointers to its parents and children in the explicit and greedy policy graphs.

- $P_n(\cdot)$ – a probability distribution on the continuous variables in node $n$. For each $\mathbf{x} \in \mathbf{X}$, $P_n(\mathbf{x})$ is an estimate of the probability density of passing through state $(n, \mathbf{x})$ under the current greedy policy. It is obtained by *progressing* the initial state forward through the optimal actions of the greedy policy. With each $P_n$, we maintain the probability of passing through $n$ under the greedy policy:

$$M(P_n) = \int_{\mathbf{x} \in \mathbf{X}} P_n(\mathbf{x}) d\mathbf{x} \ .$$

- $H_n(\cdot)$ – the heuristic function. For each $\mathbf{x} \in \mathbf{X}$, $H_n(\mathbf{x})$ is a heuristic estimate of the optimal expected reward from state $(n, \mathbf{x})$.
- $V_n(\cdot)$ – the value function. At the leaf nodes of the explicit graph, $V_n = H_n$. At the non-leaf nodes of the explicit graph, $V_n$ is obtained by backing up the $H$ functions from the descendant leaves. If the heuristic function $H_{n'}$ is admissible in all leaf nodes $n'$, then $V_n(\mathbf{x})$ is an upper bound on the optimal reward to come from $(n, \mathbf{x})$ for all $\mathbf{x}$ reachable under the greedy policy.
- $g_n$ – a heuristic estimate of the increase in value of the greedy policy that we would get by expanding node $n$. If $H_n$ is admissible then $g_n$ represents an upper bound on the gain in expected reward. The gain $g_n$ is used to determine the priority of nodes in the OPEN list ($g_n = 0$ if $n$ is in CLOSED), and to bound the error of the greedy solution at each iteration of the algorithm.

Note that some of this information is redundant. Nevertheless, it is convenient to maintain all of it so that the algorithm can easily access it. HAO* uses the customary OPEN and CLOSED lists maintained by AO*. They encode the explicit graph and the current greedy policy. CLOSED contains expanded nodes, and OPEN contains unexpanded nodes and nodes that need to be re-expanded.

## The HAO* Algorithm

Algorithm 1 presents the main procedure. The crucial steps are described in detail below.

**Expanding a node (lines 10 to 20):** At each iteration, HAO* expands the open node $n$ with the highest priority $g_n$ in the greedy graph. An important distinction between AO* and HAO* is that in the latter, nodes are often only partially expanded (i.e., not all Markov states associated with a discrete node are considered). Thus, nodes in the CLOSED list are sometimes put back in OPEN (line 23). The reason for this is that a Markov state associated with this node, that was previously considered unreachable, may now be reachable. Technically, what happens is that as a result of finding a new path to a node, the probability distribution over it is updated (line 23), possibly increasing the probability of some Markov state from 0 to some positive value. This process is illustrated in Figure 1. Thus, while standard AO* expands only tip nodes, HAO* sometimes expands nodes that were moved from CLOSED to OPEN and are "in the middle of" the greedy policy subgraph.

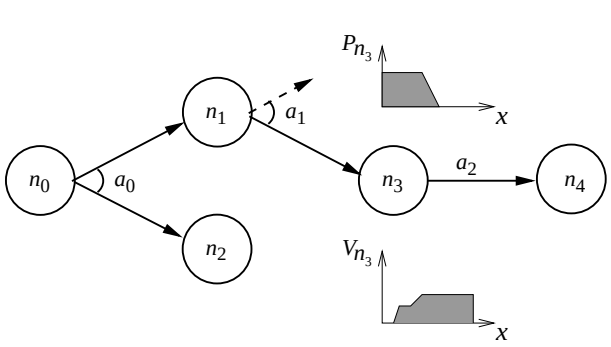
Next, HAO* considers all possible successors $(a, n')$ of $n$ given the state distribution $P_n$. Typically, when $n$ is expanded for the first time, we enumerate all actions $a$ possible in $(n, \mathbf{x})$ ($a \in A_n(\mathbf{x})$ ) for some reachable $\mathbf{x}$ ($P_n(\mathbf{x}) > 0$), and all arrival states $n'$ that can result from such a transition ($\Pr(n' \mid n, \mathbf{x}, a) > 0$).[2] If $n$ was previously expanded (i.e. it has been put back in OPEN), only actions and arrival nodes not yet expanded are considered. In line 11, we check whether a node has already been generated. This is not necessary if the graph is a tree (i.e., there is only one way to get to each discrete state).[3] In line 15, a node $n'$ is terminal if no action is executable in it (because of lack of resources). In our application domain each goal pays only once, thus the nodes in which all goals of the problem have been achieved are also terminal. Finally, the test in line 19 prevents loops in the explicit graph. As discussed earlier, such loops are always suboptimal.

1: Create the root node $n_0$ which represents the initial state.
2: $P_{n_0}$ = initial distribution on resources.
3: $V_{n_0} = 0$ everywhere in $\mathbf{X}$.
4: $g_{n_0} = 0$.
5: OPEN = GREEDY = $\{n_0\}$.
6: CLOSED = $\emptyset$.
7: **while** OPEN $\cap$ GREEDY $\neq \emptyset$ **do**
8: $n = \arg\max_{n' \in \text{OPEN} \cap \text{GREEDY}} (g_{n'})$.
9: Move $n$ from OPEN to CLOSED.
10: **for all** $(a, n') \in A \times N$ not expanded yet in $n$ and reachable under $P_n$ **do**
11: **if** $n' \notin$ OPEN $\cup$ CLOSED **then**
12: Create the data structure to represent $n'$ and add the transition $(n, a, n')$ to the explicit graph.
13: Get $H_{n'}$.
14: $V_{n'} = H_{n'}$ everywhere in $\mathbf{X}$.
15: **if** $n'$ is terminal: **then**
16: Add $n'$ to CLOSED.
17: **else**
18: Add $n'$ to OPEN.
19: **else if** $n'$ is not an ancestor of $n$ in the explicit graph **then**
20: Add the transition $(n, a, n')$ to the explicit graph.
21: **if** some pair $(a, n')$ was expanded at previous step (10) **then**
22: Update $V_n$ for the expanded node $n$ and some of its ancestors in the explicit graph, with Algorithm 2.
23: Update $P_{n'}$ and $g_{n'}$ using Algorithm 3 for the nodes $n'$ that are children of the expanded node or of a node where the optimal decision changed at the previous step (22). Move every node $n' \in$ CLOSED where $P$ changed back into OPEN.

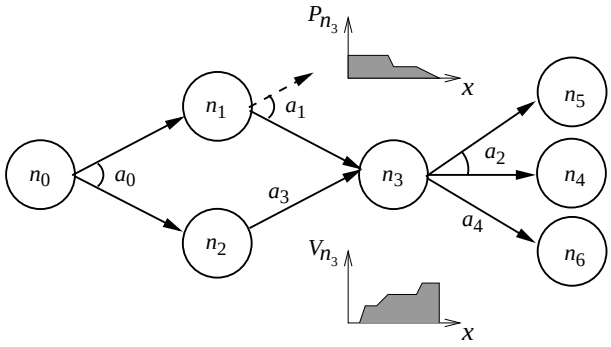
**Algorithm 1:** Hybrid AO*

[2] We assume that performing an action in a state where it is not allowed is an error that ends execution with zero or constant reward.

[3] Sometimes it is beneficial to use the tree implementation of AO* when the problem graph is *almost* a tree, by duplicating nodes that represents the same (discrete) state reached through different paths.




(a) Initial GREEDY graph. Actions have multiple possible discrete effects (e.g., $a_0$ has two possible effects in $n_0$). The curves represent the current probability distribution $P$ and value function $V$ over $x$ values for $n_3$. $n_2$ is a fringe node.




(b) GREEDY graph with $n_2$ expanded. Since the path $(n_0, n_2, n_3)$ is optimal for some resource levels in $n_0$, $P_{n_3}$ has changed. As a consequence, $n_3$ has been re-expanded , showing that node $n_5$ is now reachable from $n_3$ under $a_2$, and action $a_4$ has become do-able in $n_3$.

Figure 1: Node re-expansion.

**Updating the value functions (lines 22 to 23):** As in standard AO*, the value of a newly expanded node must be updated. This consists of recomputing its value function with Bellman's equations (Eqn. 1), based on the value functions of all children of $n$ in the explicit graph. Note that these backups involve all continuous states $\mathbf{x} \in \mathbf{X}$ for each node, *not* just the reachable values of $\mathbf{x}$. However, they consider only actions and arrival nodes that are reachable according to $P_n$. Once the value of a state is updated, its new value must be propagated backward in the explicit graph. The backward propagation stops at nodes where the value function is not modified, and/or at the root node. The whole process is performed by applying Algorithm 2 to the newly expanded node.

1: $Z = \{n\}$ // $n$ is the newly expanded node.
2: **while** $Z \neq \emptyset$ **do**
3: Choose a node $n' \in Z$ that has no descendant in $Z$.
4: Remove $n'$ from $Z$.
5: Update $V_{n'}$ following Eqn. 1.
6: **if** $V_{n'}$ was modified at the previous step **then**
7: Add all parents of $n'$ in the explicit graph to $Z$.
8: **if** optimal decision changes for some $(n', \mathbf{x})$, $P_{n'}(\mathbf{x}) > 0$ **then**
9: Update the greedy subgraph (GREEDY) at $n'$ if necessary.
10: Mark $n'$ for use at line 23 of Algorithm 1.

**Algorithm 2:** Updating the value functions $V_n$.

**Updating the state distributions (line 23):** $P_n$'s represent the state distribution *under the greedy policy*, and they need to be updated after recomputing the greedy policy. More precisely, $P$ needs to be updated in each descendant of a node where the optimal decision changed. To update a node $n$, we consider all its parents $n'$ in the greedy policy graph, and all the actions $a$ that can lead from one of the parents to $n$. The probability of getting to $n$ with a continuous component $\mathbf{x}$ is the sum over all $(n', a)$ and all possible values of $\mathbf{x}'$ of the continuous component over the the probability of arriving from $n'$ and $\mathbf{x}'$ under $a$. This can be expressed as:

$$P_n(\mathbf{x}) = \sum_{(n',a)\in\Omega_n} \int_{\mathbf{X}'} P_{n'}(\mathbf{x}') \Pr(n \mid n', \mathbf{x}', a) \Pr(\mathbf{x} \mid n', \mathbf{x}', a, n) d\mathbf{x}' \ . \quad (2)$$

Here, $\mathbf{X}'$ is the domain of possible values for $\mathbf{x}'$, and $\Omega_n$ is the set of pairs $(n', a)$ where $a$ is the greedy action in $n'$ for some reachable resource level:

$$\Omega_n = \{(n', a) \in N \times A : \exists \mathbf{x} \in \mathbf{X}, P_{n'}(\mathbf{x}) > 0,\ \mu^*_{n'}(\mathbf{x}) = a,\ \Pr(n \mid n', \mathbf{x}, a) > 0\} \ ,$$

where $\mu^*_n(\mathbf{x}) \in A$ is the greedy action in $(n, \mathbf{x})$. Clearly, we can restrict our attention to state-action pairs in $\Omega_n$, only. Note that this operation may induce a loss of total probability mass ($P_n < \sum_{n'} P_{n'}$) because we can run out of a resource during the transition and end up in a sink state.

When the distribution $P_n$ of a node $n$ in the OPEN list is updated, its priority $g_n$ is recomputed using the following equation (the priority of nodes in CLOSED is maintained as 0):

$$g_n = \int_{\mathbf{x}\in S(P_n)-\mathbf{X}_n^{\text{old}}} P_n(\mathbf{x}) H_n(\mathbf{x}) d\mathbf{x} \ ; \quad (3)$$

where $S(P)$ is the support of $P$: $S(P) = \{\mathbf{x} \in \mathbf{X} : P(\mathbf{x}) > 0\}$, and $\mathbf{X}_n^{\text{old}}$ contains all $\mathbf{x} \in \mathbf{X}$ such that the state $(n, \mathbf{x})$ has already been expanded before ($\mathbf{X}_n^{\text{old}} = \emptyset$ if $n$ has never been expanded). The techniques used to represent the continuous probability distributions $P_n$ and compute the continuous integrals are discussed

in the next sub-section. Algorithm 3 presents the state distribution updates. It applies to the set of nodes where the greedy decision changed during value updates (including the newly expanded node, i.e. $n$ in HAO* – Algorithm 1).

1: $Z$ = children of nodes where the optimal decision changed when updating value functions in Algorithm 1.
2: **while** $Z \neq \emptyset$ **do**
3: Choose a node $n \in Z$ that has no ancestor in $Z$.
4: Remove $n$ from $Z$.
5: Update $P_n$ following Eqn. 2.
6: **if** $P_n$ was modified at step 5 **then**
7: Move $n$ from CLOSED to OPEN.
8: Update the greedy subgraph (GREEDY) at $n$ if necessary.
9: Update $g_n$ following Eqn. 3.

**Algorithm 3:** Updating the state distributions $P_n$.

## Handling Continuous Variables

Computationally, the most challenging aspect of HAO* is the handling of continuous state variables, and particularly the computation of the continuous integral in Bellman backups and Eqns. 2 and 3. We approach this problem using the ideas developed in (Feng *et al.* 2004) for the same application domain. However, we note that HAO* could also be used with other models of uncertainty and continuous variables, as long as the value functions can be computed exactly in finite time. The approach of (Feng *et al.* 2004) exploits the structure in the continuous value functions of the type of problems we are addressing. These value functions typically appear as collections of humps and plateaus, each of which corresponds to a region in the state space where similar goals are pursued by the optimal policy (see Fig. 3). The sharpness of the hump or the edge of a plateau reflects uncertainty of achieving these goals. Constraints imposing minimal resource levels before attempting risky actions introduce sharp cuts in the regions. Such structure is exploited by grouping states that belong to the same plateau, while reserving a fine discretization for the regions of the state space where it is the most useful (such as the edges of plateaus).

To adapt the approach of (Feng *et al.* 2004), we make some assumptions that imply that our value functions can be represented as piece-wise constant or linear. Specifically, we assume that the continuous state space induced by every discrete state can be divided into hyper-rectangles in each of which the following holds: (i) The same actions are applicable. (ii) The reward function is piece-wise constant or linear. (iii) The distribution of discrete effects of each action are identical. (iv) The set of arrival values or value variations for the continuous variables is discrete and constant. Assumptions (i-iii) follow from the hypotheses made in our domain models. Assumption (iv) comes down to discretizing the actions' resource consumptions, which is an approximation. It contrasts with the naive approach that consists of discretizing the state space regardless of the relevance of the partition introduced. Instead, we discretize the action outcomes first, and then deduce a partition of the state space from it. The state-space partition is kept as coarse as possible, so that only the relevant distinctions between (continuous) states are taken into account. Given the above conditions, it can be shown (see (Feng *et al.* 2004)) that for any finite horizon, for any discrete state, there exists a partition of the continuous space into hyper-rectangles over which the optimal value function is piece-wise constant or linear. The implementation represents the value functions as kd-trees, using a fast algorithm to intersect kd-trees (Friedman *et al.* 1977), and merging adjacent pieces of the value function based on their value. We augmented this approach by representing the continuous state distributions $P_n$ as piecewise constant functions of the continuous variables. Under the set of hypotheses above, if the initial probability distribution on the continuous variables is piecewise constant, then the probability distribution after any finite number of actions is too, and Eqn. 2 may always be computed in finite time.[4]

## Properties

As for standard AO*, it can be shown that if the heuristic functions $H_n$ are admissible (optimistic), the actions have positive resource consumptions, and *the continuous backups are computed exactly*, then: (i) at each step of HAO*, $V_n(\mathbf{x})$ is an upper-bound on the optimal expected return in $(n, \mathbf{x})$, for all $(n, \mathbf{x})$ expanded by HAO*; (ii) HAO* terminates after a finite number of iterations; (iii) after termination, $V_n(\mathbf{x})$ is equal to the optimal expected return in $(n, \mathbf{x})$, for all $(n, \mathbf{x})$ reachable under the greedy policy ($P_n(\mathbf{x}) > 0$). Moreover, if we assume that, in each state, there is a *done* action that terminates execution with zero reward (in a rover problem, we would then start a safe sequence), then we can evaluate the greedy policy at each step of the algorithm by assuming that execution ends each time we reach a leaf of the greedy subgraph. Under the same hypotheses, the error of the greedy policy at each step of the algorithm is bounded by $\sum_{n \in \mathrm{GREEDY} \cap \mathrm{OPEN}} g_n$. This property allows trading computation time for accuracy by stopping the algorithm early.

## Heuristic Functions

The heuristic function $H_n$ helps focus the search on truly useful reachable states. It is essential for tackling real-size problems. Our heuristic function is obtained by solving a relaxed problem. The relaxation is very simple: we assume deterministic transitions for the continuous variables, i.e., $Pr(\mathbf{x}'|n, \mathbf{x}, a, n') \in \{0, 1\}$. If we assume the action consumes the minimum amount of each resource, we obtain an admissible heuristic function. A non-admissible, but probably more informative heuristic function is obtained by using the mean resource consumption.

The central idea is to use *the same algorithm* to solve both the relaxed and the original problem. Unlike classical approaches where a relaxed plan is generated for every search state, we generate a "relaxed" search-graph using our HAO* algorithm *once* with a deterministic-consumption model and a trivial heuristic. The value function $V_n$ of a node in the

[4] A deterministic starting state $\mathbf{x_0}$ is represented by a uniform distribution with very small rectangular support centered in $\mathbf{x_0}$.

relaxed graph represents the heuristic function $H_n$ of the associated node in the original problem graph. Solving the relaxed problem with HAO* is considerably easier, because the structure and the updates of the value functions $V_n$ and of the probabilities $P_n$ are much simpler than in the original domain. However, we run into the following problem: deterministic consumption implies that the number of reachable states for any given initial state is very small (because only one continuous assignment is possible). This means that in a single expansion, we obtain information about a small number of states. To address this problem, instead of starting with the initial resource values, we assume a uniform distribution over the possible range of resource values. Because it is relatively easy to work with a uniform distribution, the computation is simple relative to the real problem, but we obtain an estimate for many more states. It is still likely that we reach states for which no heuristic estimate was obtained using these initial values. In that case, we simply recompute starting with this initial state.

## Experimental Evaluation

We tested our algorithm on a slightly simplified variant of the rover model used for NASA Ames October 2004 Intelligent Systems demo (Pedersen *et al.* 2005). In this domain, a planetary rover moves in a planar graph made of locations and paths, sets up instruments at different rocks, and performs experiments on the rocks. Actions may fail, and their energy and time consumption are uncertain. Resource consumptions are drawn from two type of distributions: uniform and normal, and then discretized. The problem instance used in our preliminary experiments is illustrated in figure 2. It contains 5 target rocks (T1 to T5) to be tested. To take a picture of a target rock, this target must be tracked. To track a target, we must register it before doing the first move.[5] Later, different targets can be lost and re-acquired when navigating along different paths. These changes are modeled as action effects in the discrete state. Overall, the problem contains 43 propositional state variables and 37 actions. Therefore, there are $2^{48}$ different discrete states, which is far beyond the reach of a flat DP algorithm.

The results presented here were obtained using a preliminary implementation of the piecewise constant DP approximations described in (Feng *et al.* 2004) based on a flat representation of state partitions instead of kd-trees. This is considerably slower than an optimal implementation. To compensate, our domain features a single abstract continuous resource, while the original domain contains two resources (time and energy). Another difference in our implementation is in the number of nodes expanded at each iteration. We adapt the findings of (Hansen and Zilberstein 2001) that overall convergence speeds up if all the nodes in OPEN are expanded at once, instead of prioritizing them based on $g_n$ values and changing the value functions after each expansion.[6] Finally, these preliminary experiments do not use the sophisticated heuristics presented earlier, but the following simple admissible heuristic: $H_n$ is the constant function equal to the sum of the utilities of all the goals not achieved in $n$.

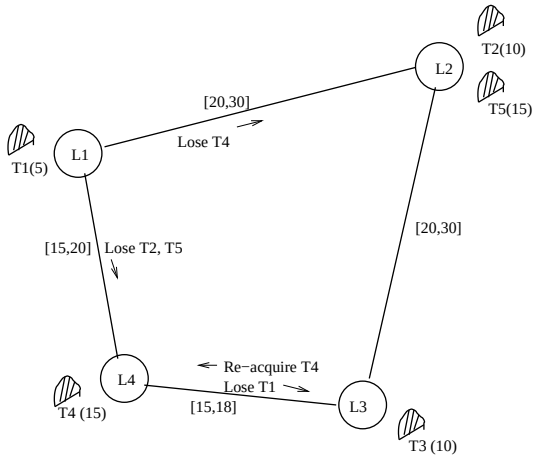



Figure 2: Case study: the rover navigates around five target rocks (T1 to T5). The number with each rock is the reward received on testing that rock.

We varied the initial amount of resource available to the rover. As available resource increases, more nodes are reachable and more reward can be gained. The performance of the algorithm is presented in Table 1. We see that the number of reachable discrete states is much smaller than the total number of states ($2^{48}$) and the number of nodes in an optimal policy is surprisingly small. This indicates that AO* is particularly well suited to our rover problems. However, the number of nodes expanded is quite close to the number of reachable discrete states. Thus, our current simple heuristic is only slightly effective in reducing the search space, and reachability makes the largest difference. This suggests that much progress can be obtained by using better heuristics. The last column measures the total number of reachable Markov states, after discretizing the action consumptions as in (Feng *et al.* 2004). This is the space that a forward search algorithm manipulating Markov states, instead of discrete states, would have to tackle. In most cases, it would be impossible to explore such space with poor quality heuristics such as ours. This indicates that our algorithm is quite effective in scaling up to very large problems by exploiting the structure presented by continuous resources.

Figure 3 shows the converged value function of the initial state of the problem. The value function is comprised of several plateaus, where different sets of goals are achieved. For example, the first plateau (until resource level 23) corresponds to the case where the resource level is insufficient for

[5] Therefore, starting to track some targets is a typical example of *set-up actions*, that is, actions that are not necessary in the nominal plan but that we must have performed before if we want to deviate from this plan, for instance, by changing goals if the current resource levels are below the expectations.

[6] In this implementation, we do not have to maintain exact probability distributions $P_n$. We just need to keep track of the supports of these distributions, which can be approximated by lower and upper bounds on each continuous variable.

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| 30 | 0.1 | 39 | 39 | 38 | 9 | 1 | 239 |
| 40 | 0.4 | 176 | 163 | 159 | 9 | 1 | 1378 |
| 50 | 1.8 | 475 | 456 | 442 | 12 | 1 | 4855 |
| 60 | 7.6 | 930 | 909 | 860 | 32 | 2 | 12888 |
| 70 | 13.4 | 1548 | 1399 | 1263 | 22 | 2 | 25205 |
| 80 | 32.4 | 2293 | 2148 | 2004 | 33 | 2 | 42853 |
| 90 | 87.3 | 3127 | 3020 | 2840 | 32 | 2 | 65252 |
| 100 | 119.4 | 4673 | 4139 | 3737 | 17 | 2 | 102689 |
| 110 | 151.0 | 6594 | 5983 | 5446 | 69 | 3 | 155733 |
| 120 | 213.3 | 12564 | 11284 | 9237 | 39 | 3 | 268962 |
| 130 | 423.2 | 19470 | 17684 | 14341 | 41 | 3 | 445107 |
| 140 | 843.1 | 28828 | 27946 | 24227 | 22 | 3 | 17113 |
| 150 | 1318.9 | 36504 | 36001 | 32997 | 22 | 3 | 1055056 |

Table 1: Performance of the algorithm for different initial resource levels. A: initial resource (abstract unit). B: execution time (s). C: # reachable discrete states. D: # nodes created by AO*. E: # nodes expanded by AO*. F: # nodes in the optimal policy graph. G: # goals achieved in the longest branch of the optimal solution. H: # reachable Markov states.

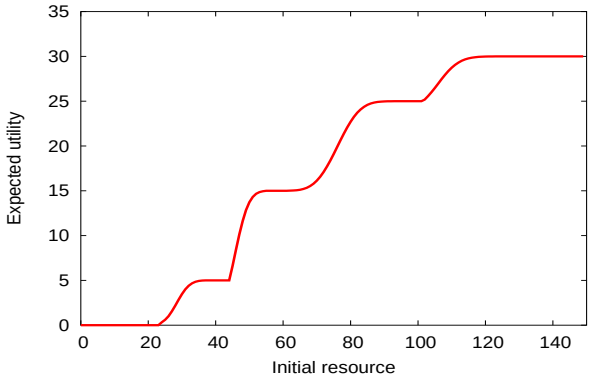



Figure 3: Value function of the initial state.

any goal to be achieved. The next plateau (until 44) depicts the region in which the target T1 is tested. The remaining resources are still not enough to move to a new location and generate additional rewards. In the region between 44 and 61 the rover decides to move to L4 and test T4. Note that the location L2 is farther from L4 and so the rover does not attempt to move to L2, yet. The next plateau corresponds to the region in which the optimal strategy is to move to L2 and test both T2 and T5, as enough resources for that are now available. The last region (beyond 101) is in which three goals T1, T2 and T5 are tested and reward of 30 is obtained.

When $H_n$ is admissible, we can bound the error of the current greedy graph by summing $g_n$ over fringe nodes. In Table 2 we describe the time/value tradeoff we found for this domain. On the one hand, we see that even a large compromise in quality leads to no more than 25% reduction in time. On the other hand, we see that much of this reduction is obtained with a very small price ($\epsilon = 0.5$). Additional experiments are required to learn if this is a general phenomenon.

| Initial resource | $\varepsilon$ | Execution time | # nodes created by AO* | # nodes expanded by AO* |
|---|---|---|---|---|
| 130 | 0.00 | 426.8 | 17684 | 14341 |
| 130 | 0.50 | 371.9 | 17570 | 14018 |
| 130 | 1.00 | 331.9 | 17486 | 13786 |
| 130 | 1.50 | 328.4 | 17462 | 13740 |
| 130 | 2.00 | 330.0 | 17462 | 13740 |
| 130 | 2.50 | 320.0 | 17417 | 13684 |
| 130 | 3.00 | 322.1 | 17417 | 13684 |
| 130 | 3.50 | 318.3 | 17404 | 13668 |
| 130 | 4.00 | 319.3 | 17404 | 13668 |
| 130 | 4.50 | 319.3 | 17404 | 13668 |
| 130 | 5.00 | 318.5 | 17404 | 13668 |
| 130 | 5.50 | 320.4 | 17404 | 13668 |
| 130 | 6.00 | 315.5 | 17356 | 13628 |

Table 2: Complexity of computing an $\varepsilon$-optimal policy. The optimal return for an initial resource of 130 is 30.

## Conclusions

We presented a variant of the AO* algorithm that, to the best of our knowledge, is the first algorithm to deal with: limited continuous resources, uncertainty, and oversubscription planning. We developed a sophisticated reachability analysis involving continuous variables that could be useful for heuristic search algorithms at large. Our preliminary implementation of this algorithm shows very promising results on a domain of practical importance. We are able to handle problems with $2^{48}$ discrete states, as well as a continuous component.

In the near future, we hope to report on a more mature version of the algorithm, which we are currently implementing. It includes: (1) a full implementation of the techniques described in (Feng *et al.* 2004); (2) a rover model with two continuous variables; (3) a more informed heuristic function, as discussed in Section 3.

### Acknowledgements

This work was funded by the NASA Intelligent Systems program. Eric Hansen was supported in part by NSF grant IIS-9984952, NASA grant NAG-2-1463 and a NASA Summer Faculty Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the NSF or NASA. This work was performed during Mausam and Eric Hansen's visit at NASA Ames Research Center.

## References

E. Altman. *Constrained Markov Decision Processes*. Chapman and HALL/CRC, 1999.

J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 77–84, 2002.

Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the Twentieth Confer-*

*ence on Uncertainty in Artificial Intelligence*, pages 154–161, 2004.

J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Mathematical Software*, 3(3):209–226, 1977.

C. Guestrin, M. Hauskrecht, and B. Kveton. Solving factored MDPs with continuous and discrete variables. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 235–242, 2004.

E. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

N. Meuleau, R. Dearden, and R. Washington. Scaling up decision theoretic planning to planetary rover problems. In *AAAI-04: Proceedings of the Workshop on Learning and Planning in Markov Processes Advances and Challenges*, pages 66–71, Technical Report WS-04-08, AAAI Press, Menlo Park, CA, 2004.

J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

L. Pedersen, D. Smith, M. Deans, R. Sargent, C. Kunz, D. Lees, and S.Rajagopalan. Mission planning and target tracking for autonomous instrument placement. In *Submitted to 2005 IEEE Aerospace Conference*, 2005.

D. Smith. Choosing objectives in over-subscription planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 393–401, 2004.

M. van den Briel, M.B. Do R. Sanchez and, and S. Kambhampati. Effective approaches for partial satisfation (oversubscription) planning. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 562–569, 2004.

H.L.S. Younes and R.G. Simmons. Solving generalized semi-Markov decision processes using continuous phase-type distributions. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 742–747, 2004.

# Symbolic Heuristic Policy Iteration Algorithms for Structured Decision-theoretic Exploration Problems

**F. Teichteil-Königsbuch** and **P. Fabiani**
ONERA-DCSD
2 Avenue Édouard-Belin
31055 Toulouse, FRANCE
(florent.teichteil,patrick.fabiani)@cert.fr

## Abstract

The *ReSSAC* project, an autonomous exploration rotorcraft project at ONERA, motivates our research on decision-theoretic planning in large state spaces. Recent work on Markov Decision Processes make it possible to address more realistic stochastic planning problems, thanks to factored models and implicit state representations. Decomposition and factorization techniques enable to plan using state and action variables. Decision-theoretic exploration problems comprises several intermediate goals and are structured in two components. A graph of enumerated states represents the navigation component of the problem, as in gridworld MDPs. A set of state variables describes, in a compact and implicit way, the other features of the problem, including the intermediate goals to be achieved in sequence. The solution of an academic instance of an exploration problem is presented. A family of gridworld exploration problems is used to compare Heuristic Search Dynamic Programming algorithms on such large scale MDPs. A common algorithmic scheme is used to compare *LAO** with a sub-optimal *Symbolic Focused Dynamic Programming (SFDP)* policy iteration algorithm. *SFDP* quickly finds sub-optimal solutions when *LAO** cannot tackle the problem. The originality of *SFDP* is that both the optimization time and the solution quality are controlled by planning for a partial subset of selected planning goals. An even faster version *sfDP* is presented that quickly finds solution in larger problems. The incremental version *IsfDP* incrementally improves the current solution thanks to iterative calls of *sfDP* with an increasing list of planning subgoals to be taken into account. We compare all these algorithms on a set of problems of different sizes.

## Introduction

The approach and the algorithms presented in this paper are developed for decision-theoretic planning in large state spaces. This work is motivated by an application to planning under uncertainty for autonomous exploration or "search and rescue" aircraft within the *ReSSAC* [1] project at ONERA.

Markov Decision Processes (MDPs) (Puterman 1994) are a reference framework for sequential decision making under uncertainty: in order to deal with the uncertain effects of the agent's actions, a *policy* is computed on the state space. It is a function giving, for every enumerated possible state, the action to be performed next. The *optimal policy* maximizes the probability of success, or the mathematical expectation of reward: the *value function* defined on every state. Classical stochastic dynamic programming algorithms are based on an explicitly enumerated and unstructured state space. The size of the state space is an exponential function of the number of features that describe the problem. The state enumeration itself may rapidly become intractable for realistic problems. More generally (Boutilier, Dean, & Hanks 1999) provide an extensive discussion on complexity and modeling issues. (Boutilier, Dearden, & Goldszmidt 2000) show the benefits of factored representations in order to avoid state enumeration, to reason at a higher level of abstraction as in (Dearden & Boutilier 1997) and to take into account non-Markovian aspects of the problem, such as historic dependent rewards or goals as shown in (Bacchus, Boutilier, & Grove 1997). Other approaches, as in (Dean & Lin 1995), introduce a state space hierarchical decomposition into sub-regions. Local policies are then computed in each sub-region and become the macro-actions applicable in the macro-states of a global abstract and factored MDP. (Teichteil & Fabiani 2005) propose to combine both decomposition and factorization techniques. Other important contributions, such as the SPUDD library (Hoey *et al.* 2000), have improved the efficiency of factored MDP solution algorithms, using decision diagrams, borrowed from the Model Checking community. Recently, heuristic search schemes have been proposed such as *LAO** (Hansen & Shlomo 2001), or *LRTDP* (Bonet & Geffner 2003b).

*Symbolic Focused Dynamic Programming (SFDP)* heuristic search algorithm is an original contribution of ours. *SFDP* conforms, like *LAO**, with a two-phased scheme of *planning space expansion*-and-*optimization*. The *planning space expansion* is based on a reachability analysis using the current policy and *planning goals*. The *optimization* stage is a dynamic programming phase applied within the previously expanded planning space: this actually enables to *focus more or less* the search and thus to control the optimization process. This enables to quickly find a first solution that is later improved if time is available. In the following, we first present our motivations for symbolic heuris-



[1] http://www.cert.fr/dcsd/RESSAC

tic decision-theoretic planning : we present the family of gridworld exploration problems. Our approach exploits the problem structure in terms of both decomposition and factorization (see also (Teichteil & Fabiani 2005)) : issues about exploration planning problems naturally motivate the decomposition of the navigation space on the one hand and the use of a factored model of the mission variables on the other hand. We propose a common heuristic scheme for the solution of decision-theoretic exploration problems. Our work is inspired by *LAO** (Hansen & Shlomo 2001) and *LRTDP* (Bonet & Geffner 2003b) value iteration algorithms. We describe our implementations and comparisons of the *policy* and the *value iteration SFDP* algorithms. The scalability of *SFDP* is presented through performance comparisons with *LAO** and *SPUDD*. We implemented a *policy iteration SPUDD* together with symbolic *value* and *policy iteration LAO**. A suboptimal faster *sfDP* version of *SFDP* is presented. An incremental version *IsfDP* iteratively calls *sfDP* in order to incrementally improve the solution. No optimal version of the *SFDP* scheme is presented here. Experiments were conducted on the same family of gridworld exploration-like problems. We conclude about further work on Heuristic Search Dynamic Programming algorithms.

## Motivations

Our research is motivated by an application to exploration or "search and rescue" autonomous aircraft. We aim at controlling the optimization process, the solution quality and the optimization time, through the enforcement of specific goals to be achieved with maximum priority. A solution should be rapidly available in case of replanning, to be further improved if time is available, up to the optimal solution. On the other hand, some specific subgoals of the problem may require to be imposed in the solution whatever happens with the other subgoals.

An exploration mission comprises a problem of navigation (see Figure 1) in a partially known environment on the one hand, and a problem of online information acquisition and replanning on the other hand. Several final, alternative or intermediate goals may be imposed to the agent, either as alternative choices (the agent may reach its final goal $O_f$ either via the prior achievement of $O_1$ or through $O_2$). Some final goals, such as landing in a safe area, must be achieved in all possible plans. Some goals, such as exploring or searching a region, are the pre-conditions to be achieved before seeking to realize further goals. Some of the agent's rewards and goals are historic dependent, and there may exist ordering constraints between the goals, e.g. the agent must take information in region $R_k$ and transmit it to its ground control center before proceeding with its navigation to the neighboring regions. Other state variables, such as the agent's energy autonomy level $A$, are clearly orthogonal to the navigation components, yet not at all decoupled from it: each aircraft move consumes some energy, which is modelled as a probability of transition to a lower value of the energy autonomy variable. An insufficient energy autonomy level can force the agent to *Abort* its mission and return to its base, or to land on an emergency or security crash base. Another stochastic variable of the problem concerns the aircraft vertical distance above the ground (and obstacles) : considering that the terrain is partially known, the aircraft's action of going to a waypoint may result in different resulting vertical distance to the ground and obstacles at that point. There are constraints on the possible vertical motions of the aircraft, such that transitions from "cruise altitude" to "landing" can only be achieved through intermediate "approach altitude" at which terrain observation and exploration can be performed prior to a safe "landing". Emergency landing may result in a direct transition to the ground level, but with associated possible damages. Similarly, the airspeed of the rotorcraft must be adapted to the vertical distance to the ground and obstacles in order to avoid damages. Different possible actions of transition between the possible waypoints can be generated either by optimizing the transition time, the energy consumption or the achievement of some perception tasks.

Last but not least, it may be specified, as in our simple example, that rewards can only be obtained once: so that having achieved goal $O_1$ in region $R_1$ nullifies the corresponding reward and thus completely changes the optimal strategy in that region. Computing a strategy for each possible combinations of goals $O_j$ being achieved or not, leads to perform a number of solution that is exponential in the number of such possible intermediate goals. Ordering constraints could also be imposed on the sequence of achievement of some sub-goals.

Such a problem is not Markovian. This context is precisely addressed in (Bacchus, Boutilier, & Grove 1997), where a temporal logic is used to describe and reason on such constraints and generate the required "additional variables" to take this into account. Today, we introduce these additional state variables "by hand" in the problem imperfectly sketched on Figure 1. We encode the above constraints using *Dynamic Bayes Nets (DBNs)* (Dean & Kanazawa 1989) as shown in Figure 3.

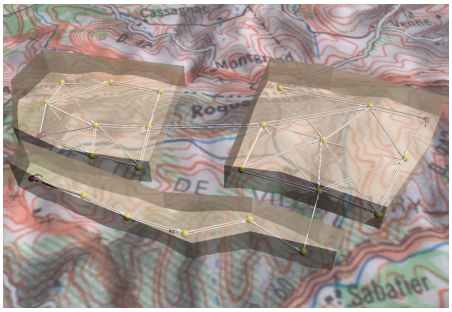


Figure 1: Navigation component of the problem : 3 regions and 3 rewards that can be obtained once and in turn

Throughout this paper, we use a family of similar gridworld problems of different sizes and complexity, with weakly coupled regions and additional state variables, especially variables that describes which goals have been achieved or not.

Gridworlds are used because they make it easier to generate large scale problems as in (Bonet & Geffner 2003b).

Simple problems are used to explain the approach and more complex ones to demonstrate the scalability.

## Markov Decision Processes (MDPs)

A MDP (Puterman 1994) is a Markov chain controlled by an agent. A control strategy $\pi$ that associates to each state the choice of an action is called a *policy*. Each action triggers a stochastic transition to neighbor states. The Markov property means that the probability of arriving in a particular state after an action only depends on the previous state of the chain and not on the entire states history. Formally it is a tuple $\langle S, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where $S$ is the set of possible states, $\mathcal{A}$ is the set of actions, $\mathcal{T}$ and $\mathcal{R}$ are respectively the transition probabilities and rewards functions of the MDP.

$$\forall\ s, a, s' \in S \times \mathcal{A} \times S\ ,\ \left\{ \begin{array}{l} T(s,a,s') = T(s'|a,s) \\ R(s,a,s') = R(s'|a,s) \end{array} \right.$$

$\mathcal{T}$ and $\mathcal{R}$ values depend on the starting state, the ending state and the chosen action (probabilities and rewards are stored in matrices or tables).

The most frequently used optimization criterion consists in maximizing the value function $V^\pi = E\left(\sum_{t=0}^{\infty} \beta^t\, r_t^\pi\right)$, i.e. the infinite sum of expected rewards $r_t^\pi$ obtained while applying policy $\pi$, discounted by a factor $0 < \beta < 1$. $\beta$ insures the sum's convergence, but can also be interpreted as an uncontrolled stopping probability (system failure or mission end) between two time points. Two main classes of iterative dynamic programming algorithms are classically applied to MDPs, based on Bellman equations (Bellman 1957) for the value $V^\pi$ of a policy $\pi$ :

$$V^\pi(s) = \sum_{s' \in S} T(s, \pi(s), s') \cdot \big(R(s, \pi(s), s') + \beta\, V^\pi(s')\big) \quad (1)$$

The *Value Iteration* scheme consist in computing the optimal value of $V$ by successive approximations. The proof of convergence is a fixed point theorem: a unique optimal $V^*$ is reached when $V$ stabilizes. $V_{k+1}$ is computed on each state $s$ at iteration $k+1$ as the maximum achievable value assuming that $V_k$ is then obtained in any state $s'$ that is reached by applying an action $a$ from state $s$:

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in S} T(s,a,s').\big(R(s,a,s') + \beta.V_k(s')\big) \quad (2)$$

The *Policy iteration* scheme requires to assess $V_{\pi_k}$ for the current policy $\pi_k$: approximate solutions of equation 1 are obtained by linear programming, or successive approximations techniques. $\pi_{k+1}$ is then the greedy policy based on $V_{\pi_k}$:

$$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in S} T(s,a,s').\big(R(s,a,s') + \beta.V_{\pi_k}(s')\big) \quad (3)$$

Compared to value iteration, the policy iteration algorithm converges in fewer iterations, but each policy assessment stage may be computationally costly. A large discussion about criteria and resolution algorithms is proposed in (Puterman 1994).

## Decomposition of our exploration MDP

The decomposition of an MDP is based on a state space partition $\Pi$ into non empty distinct *regions*. Enumerated states are thus grouped into weakly coupled regions, i.e. with fewer transitions between the exit and entrance states of the regions. An abstract MDP (Dean & Lin 1995) is then constructed, on the basis of *macro-states* resulting from the decomposition into regions. For each region $r$, $Sper(r)$ is the set of the exit states of the region $r$ :

$$Sper(r) = \big\{ s \in S - r \ /\ \exists\ s' \in r\ ,\ \exists\ a \in A\ ,\ T(s',a,s) \neq 0 \big\}$$

We found two main possible options for the definition of macro-states. In (Hauskrecht *et al.* 1998), macro-states are the exit states of the regions, i.e. the state space of the abstract MDP is $\bigcup_{r \in \Pi} Sper(r)$. In (Dean & Lin 1995), macro-states are the aggregate states of the regions. The first model leads to possibly sub-optimal policies, because it only considers strategies leading from a macro-state to a different macro-state. This model does not allow a local strategy to try and reach a local sub-goal by staying within the same macro-states (absorbing macro-states are not possible). This aspect of the problem leads us to choose the second model for our exploration problem (like in Figure 1): each region possibly contains a local sub-goal to be achieved by a specific locally optimal strategy before leaving the region with another strategy. For our navigation MDP in Figure 1, we obtain the kind of decomposition shown in Figure 2.

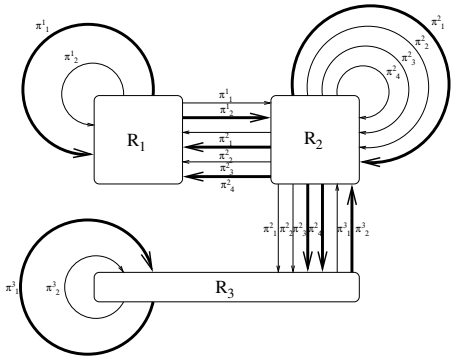



Figure 2: Example of abstract MDP (from figure 1)

More generally, an abstract MDP (Dean & Lin 1995) is a tuple $\langle S', \mathcal{A}', \mathcal{T}', \mathcal{R}' \rangle$. The actions of the abstract MDP are then the local policies generated in each region and the transitions in the abstract MDP are computed from these local policies:

- $S' = \bigcup_{r \in \Pi} r$,
- $\mathcal{A}' = \bigcup_{r \in \Pi} \big\{ \pi_1^r, \ldots \pi_{m_r}^r \big\}$,
- $T'(r, \pi_j^r, r')$ and $R'(r, \pi_j^r, r')$ depend on a *discounted macro-transition model*.

Though our decomposition model is borrowed from (Dean & Lin 1995), our decomposition algorithm is based on the techniques proposed by (Hauskrecht *et al.* 1998) because they are more adapted to factored resolution algorithms. In order to adapt a coherent *discounted macro-transition model*, we simply applied the computation methods borrowed from (Hauskrecht *et al.* 1998) and matched it on

the macro-state decomposition borrowed from (Dean & Lin 1995). As a result, in our implementation, the macro-probability of transition from $r$ to $r'$ applying macro-action $\pi_j^r$ is computed as the average probability of moving from somewhere in a region $r$ (assuming a uniform probability distribution on starting states in $r$) to an entrance state of a region $r'$. The macro-reward for the same macro-action and the same transition is computed as the exactly corresponding average reward.

- $T(r, \pi_j^r, r') = \frac{1}{|r|} \sum_{s' \in Sper(r) \cap r'} \sum_{s \in r} T(s, \pi_j^r, s')$
- $R(r, \pi_j^r, r') = \frac{1}{|r|} \sum_{s' \in Sper(r) \cap r'} \sum_{s \in r} T(s, \pi_j^r, s') \cdot R(s, \pi_j^r, s')$

$T(s, \pi_j^r, s')$ and $R(s, \pi_j^r, s')$ are iteratively computed like the value function in a Gauss-Seidel value iteration algorithm (Puterman 1994) (see also (Teichteil & Fabiani 2005)).

As a result, the proofs in (Hauskrecht *et al.* 1998) still apply in our case with respect to local policies. In each macro-state, local policies are optimized for local MDPs, that include the local states of the macro-state itself, in our case $r \cup Sper(r)$, and an absorbing state $\alpha$, connected to each exit state $s \in Sper(r)$ and standing for "the rest of the world" from the viewpoint of the local MDP. Special reward functions are defined on the transitions from the exit states in $Sper(r)$ to these absorbing states (called *peripheral values*) that correspond to the total sum of expected rewards for the agent if it escapes the local MDP, averaged on all what can happen "outside the local MDP". This is why $\alpha$ is an absorbing state. This is also why the value of reaching $\alpha$ is not straightforward to compute, and why $Sper(r)$ has to be included in the *macro-state*. Let the reward obtained in the transition from exit state $s \in Sper(r)$ to the absorbing state $\alpha$ be $\lambda(s)$: note that $V(s) = \lambda(s)$. However, it is also clear that each possible combination of the *peripheral values* $\lambda(s)$ for all $s \in Sper(r)$ will change the obtained optimal local policy. As a result, *peripheral values* must be taken as parameters.

Our local MDPs $\langle S', A', T', R' \rangle^{(\lambda(s))_{s \in Sper(r)}}$ are then defined, for each region $r$ and for each as follows:

- $S' = r \cup Sper(r) \cup \{\alpha\}$ , where $\alpha$ is an absorbing state,
- $A' = A$ ,
- $T(s, a, s') = \begin{cases} T(s, a, s') & \text{if } (s, s') \in r \times (r \cup Sper(r)) \\ 1 & \text{if } (s, s') \in (Sper(r) \cup \{\alpha\}) \times \{\alpha\} \end{cases}$
- $R(s, a, s') = \begin{cases} R(s, a, s') & \text{if } (s, s') \in r \times (r \cup Sper(r)) \\ \lambda(s) & \text{if } (s, s') \in Sper(r) \times \{\alpha\} \\ 0 & \text{if } (s, s') \in \{\alpha\}^2 \end{cases}$

For each combination of peripheral values, there is an optimal local policy. However, a same local policy can be optimal with different combinations of peripheral values. As a result, for each local MDP, we need to generate a set of local policies that should be as small as possible, but still exhaustive since at least one local policy per region should match the global optimal policy restricted to this region.

The following **theorem** is proved in (Hauskrecht *et al.* 1998): *for any local policy $\pi_r^*$ on a local MDP, if it is locally optimal for a combination of peripheral values $(\lambda(s))_{s \in Sper(r)}$ that corresponds to the actual optimal value function of the global MDP restricted to the corresponding exit states in $Sper(r)$, then it matches the optimal policy $\pi^*$ of the global MDP in this region.*

A first resolution method (Hauskrecht *et al.* 1998), named *coverage technique* (CT), consists in constructing a mesh of peripheral values covering an interval $[V^*_{min}, V^*_{max}]$ and whose spacing $\delta$ is chosen such that the value of a local policy is within $\frac{\delta}{2}$ of the optimal policy value. This method requires to set bounds on the optimal value function $V^*$, which is generally not a problem. It is however rather ineffective in the sense that it requires to generate at least $\prod_{s \in Sper(r)} \frac{V^*_{max}(s) - V^*_{min}(s)}{\delta}$ grid points in total, with as many resolutions of the local MDP. Besides, the corresponding resolutions are mostly redundant: a same optimal local policy can be obtained through different resolutions and the method even generates redundant dominated policies of equivalent local value that need to be eliminated.

A second resolution method is proposed in (Parr 1998) which is based on *Linear Programming* (LP). For each local policy $\pi_r$ on the macro state $r \cup Sper(r)$, the value function on the internal states $s \in r$ is a linear combination of the values on the exit states $s \in Sper(r)$, i.e. $\lambda(s)$. Thus, the dominating policies at any state $s$ form a piecewise-linear convex function of the peripheral values, so that the necessary and sufficient set of local policies that are optimal for each combination of peripheral values can be generated (Parr 1998) with methods inspired from the resolution of Partially Observable MDPs (Cassandra 1998). Yet no numerical results are provided in (Parr 1998).

We thus had to compare experimentally these two methods on our exploration problems: numerical results obtained on problems similar to Figure 1 are presented in (Teichteil & Fabiani 2005). The LP method appears to be the most effective. First, the complexity of LP is polynomial in the number of exit states whereas the one of CT is exponential. Second, CT produces useless policies that are dominated by the others contrary to LP policies that are all non dominated policies. Third, CT generates more than $99\%$ policies that have the same values and that must be also eliminated. Fourth, CT sometimes constructs less policies than LP because some optimal policies for given peripheral values on exit states lie between two points of the grid. Therefore, we have implemented the LP technique in our approach in order to decompose the navigation state sub-space into regions and compute navigation macro-actions for our factored abstract global exploration MDP.

On our simple instance, LP produced respectively 7, 6 and 2 local policies respectively in regions $R_1$, $R_2$ and $R_3$. The computed macro-transitions for each generated local policy for region $R_1$ are shown in Table 1. Three local policies (one in each region) are actually depicted in Figure 1. The probabilities of the macro-transitions in Table 1 give a good idea of the local behaviors.

## Decomposition of our simple instance

The decomposition of the problem is motivated by optimization time concerns. Work by (Teichteil & Fabiani 2005) propose to factorize such problems through a hierarchical

| Origin region | Local policies | End region | Probability | Value |
|---|---|---|---|---|
| $R_1$ | $\pi_1^1$ | $R_1$ | 1 | 17.6327 |
| | $\pi_2^1$ | $R_1$ | 0.111317 | 2.21295 |
| | | $R_2$ | 0.888683 | 0 |
| | $\pi_3^1$ | $R_1$ | 0.211306 | 4.00467 |
| | | $R_2$ | 0.788694 | 0 |
| | $\pi_4^1$ | $R_1$ | 0.433556 | 8.18566 |
| | | $R_2$ | 0.566444 | 0 |
| | $\pi_5^1$ | $R_1$ | 0.666778 | 11.5589 |
| | | $R_2$ | 0.333222 | 0 |
| | $\pi_6^1$ | $R_1$ | 0.988889 | 17.1386 |
| | | $R_2$ | 0.0111111 | 0 |
| | $\pi_7^1$ | $R_1$ | 0.446756 | 8.21195 |
| | | $R_2$ | 0.553244 | 0 |

Table 1: Local policies and macro-transitions for region $R_1$

decomposition into an abstract factored MDP, and show the benefits of it in terms of efficiency. (Hauskrecht *et al.* 1998) or (Parr 1998) have proposed two candidate algorithms for the computation of local policies during the MDP decomposition step. According to (Teichteil & Fabiani 2005) the latter approach (*LP* by (Parr 1998)) based on linear programming, is the one that offers the better performances and flexibility. In order to take into account the fact that rewards can only be obtained once, we have to adapt R. Parr (1998) algorithm to our problem. We need to optimize the regions local policies conditionally to the status of the goals of the region: in practice this limits greatly the number of cases since the combinatorics is splitted into the regions. In our simple example, we only have one goal per region, which leads to optimize 2 sets of *conditional local policies* per region : one if the local *goal* has not been achieved yet by the agent, and one if it has already been. The direct benefit driven from decomposition comes from the fact that if there are $k$ regions and one goal per region, only $2k$ local policies are computed. Without decomposition $2^k$ global policies should be optimized. After the decomposition phase, the navigation graph component of the gridworld exploration problem is splitted into *macro-states*, each one corresponding to a sub-region (see Figure 2, and combined with the other orthogonal state variables. The resulting abstract MDP is in a factored form and may be represented by Dynamic Bayes Nets as in Figure 3. The state variable $R$ stands for the region, $O_1$, $O_2$ and $O_3$ stand for the *goals*, and $A$ for the agent energy autonomy level. For simplicity, we assumed a binary energy autonomy level with constant consumption over the regions, the function $f$ giving the probability of "loosing the minimal energy autonomy level" between two time steps: $f(R_i, R_j, \pi) = [\ 0.65\ 0.35\ ]$ for all $i$,$j$ and $\pi$.

## Algebraic Decision Diagrams

Our concern about encoding efficiency is related to the optimization time issue in our research context. In DBNs, the transition probabilities and rewards are represented by *Conditional Probability Tables*, i.e. large matrices, one for every post-action variables. However, these probability and reward tables are sparse in most problems and can be encoded in "decision trees" as in (Boutilier, Dearden, & Goldszmidt 2000), or as "algebraic decision diagrams" (ADDs) as in (Hoey *et al.* 2000). Figure 4 shows instances of transition probability trees (vectors at the leafs) and transition rewards trees for the macro-actions of our instance of abstract MDP. For each post-action variable state, every leaf of the probability tree stores a list containing the probabilities to obtain every possible value $x_i^{t+1}$ of this variable, knowing the values of the other variables $x_i^t$, $x_j^t$, $x_j^{t+1}$ along the path $x_i^t \wedge \left(\wedge_{j \neq i} \left(x_j^t \wedge x_j^{t+1}\right)\right)$ from the root of the tree to the considered leaf. The reward tree on Figure 4 expresses the fact that, in order to obtain the reward associated with a given goal, this goal must not be already achieved and the agent must first reach the corresponding region with a sufficient energy autonomy level. Goals in the other regions cannot be achieved from "outside" and the corresponding decision tree is equivalent to a *NO-OP*. ADDs offer the additional advantage that nodes of a same value are merged, thus leading to a graph (see Figure 5) instead of a tree. After testing and comparing Decision Trees and ADDs implementations of our policy and value iteration algorithms, the conclusion was that ADDs offer a much more efficient encoding, even if they are limited to use binary conditions: some state or action variables may have to be splitted into

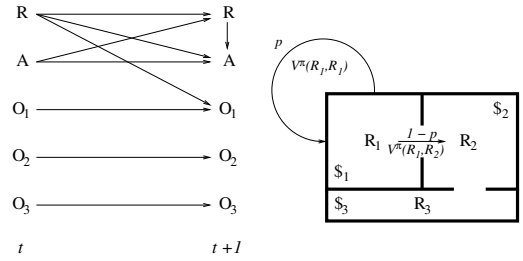



Figure 3: Action networks and transitions of the abstract MDP with the local policies of the region $R_1$

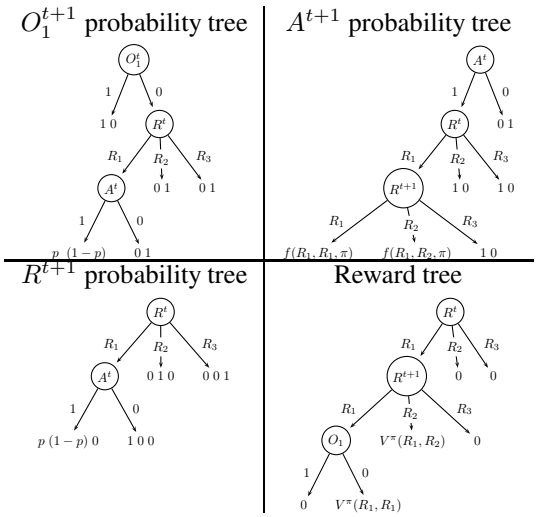



Figure 4: $O_1^{t+1}$, $A^{t+1}$ and $R^{t+1}$ probability trees, and reward tree (from $R_1$)

several binary variables to fit in the model. As a matter of fact, state variables with large range of values considerably increase the computation time necessary in order to solve factored MDPs. This is either due to the width of the corresponding trees when using Decision Trees, or otherwise due to the number of binary variables required when using ADDs. It is moreover noticeable that position or navigation variables typically take a large number of possible values. This is another way of getting convinced that it is a good idea to decompose the navigation component in our exploration problem into fewer more abstract aggregate regions. Algorithms for the solution of factored MDPs using ADDs are based on simple algebraic and logical operations such as AND, OR, PRODUCT, SUM, etc. Some technicalities specific to ADDs are explained in (Hoey *et al.* 2000), especially with respect to the implementation of *value iteration* in *SPUDD*, on which our own *value iteration* algorithms are based. The development of the *policy iteration* versions of the compared algorithms demanded to apply some similar technicalities: in order to improve the efficiency of our algorithms, we apply for each action a mask BDD on the complete action diagram ADD and the reward ADD of the action, representing the states where the action can be applied. Furthermore, BDDs and ADDs are restricted to the current reachable state space in order to save memory and to speed up the ADD operations.

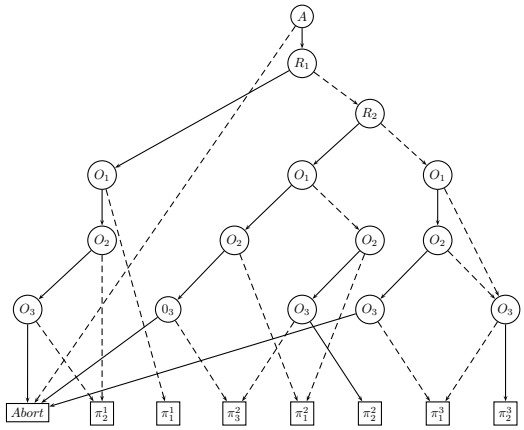



Figure 5: Optimal policy ADD of the MDP of Figure 3

## Symbolic Heuristic Search

Symbolic heuristic search dynamic programming algorithms seem to conform to a common two-phased scheme, shown in Figure 6:

- a first **reachability analysis** and **heuristic** computation phase,
- a subsequent **dynamic programming** phase.

It constitutes our common algorithmic basis for developing and comparing different heuristic search dynamic programming algorithms that conform to it such as *sLAO** (Feng & Hansen 2002) and *sRTDP* as in (Feng, Hansen, & Zilberstein 2003).

**Init**
$R_0 \leftarrow Reachable(I, \mathcal{A}, G)$
$\Pi_0 \leftarrow ShortestStochasticPath(R_0 \rightarrow G)$
$S_0 \leftarrow FilterStates(R_0, P(s) < \epsilon \cdot P(I))$
$\Longrightarrow (\Pi_0, V_0, S_0)$

$k \leftarrow 0$
**repeat**
$S_{k+1} \leftarrow Reachable(I, S_k, \Pi_k, G)$
$DynamicProgramming(\Pi_k, V_k, S_{k+1})$
$k \leftarrow k + 1$
**until** convergence over $S_k$

Figure 6: Heuristic Search Dynamic Programming Scheme

### Reachability analysis

One strong idea is simply that the algorithm cannot apply dynamic programming on the full state space because this is intractable. The working space is thus carefully extended at each iteration, keeping it small but still sweeping the full state space. **Heuristic** computations, such as the proposed *shortest stochastic path analysis* intervene in this first phase, essentially to provide an admissible estimation of the value function or equivalently, a default policy on the border of the planning space. The "planning space expansion" phase enables to control the optimization process. *sLAO** incrementally augments its working space until all the pertinent rewards have been collected and the processes converges.

**Deterministic reachability analysis** We call $Reachable(I, \Pi_A, Stop)$ a function that takes as inputs the sets of initial and goal states $I$ and $G$, uses the set of applicable actions $\Pi_A \subset \mathcal{A}$ ($\Pi_A$ can be a policy or $\mathcal{A}$ itself) and computes the set $R_0$ of all the states that are reachable from $I$ with successive applications of deterministic actions in $\mathcal{A}$ in an iterative loop that stops as soon as the $Stop$ condition is reached : e.g. $Stop$ can be $G \subset R_0$ or $1\,step\,lookahead$. The actions are made deterministic by setting the maximum transition probability to 1 and the other one to 0, which enables us to convert the ADDs into BDDs (Binary Decision Diagrams) that are more efficient. The idea here is that the planning space expansion is controlled via the $Stop$ condition, linked to the achievement of specific planning goals.

**Shortest stochastic path analysis** At this stage, we can at the same time compute an initial heuristic policy (or value function) and reduce – if possible – the initial reachable state space. We call $ShortestStochasticPath(R_0 \rightarrow G)$ a function that takes $R_0$ and $G$ as inputs and computes a shortest stochastic path from every state in $R_0$, using stochastic actions from $\mathcal{A}$ without their rewards. Better simplification schemes should certainly be studied, but this heuristic seems efficient in many problems, such as navigation grid MDPs in (Hansen & Shlomo 2001) and in (Bonet & Geffner 2003b). We call $FilterStates(R_0, P(s) < \epsilon \cdot P(I))$ a filtering function that filters the states that have a very low probability of reachability when the non-deterministic actions are applied along the shortest path trajectories. Low probability of reachability is assessed compared to the probability

of the initial states. This stage is represented in Figure 7. Stochastic reachability filtering seems very comparable in its results, with the effect of random sampling in *LRTDP* (Bonet & Geffner 2003b).

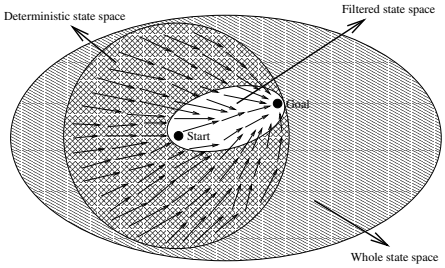



Figure 7: *SFDP focuses* when no goal constraints are set

## SFDP and sLAO*

During that stage the solution of the MDP is optimized on the current reachable state space . The expansion of the current reachable state space results from the function $Reachable$, applied either from $I$ (*SFDP*) or from the previous reachable state space (*sLAO**). The $Reachable$ function expands the set of reachable states until **all the states that satisfy the goal conditions** are reached. The $DynamicProgramming$ function of the main algorithm (see Figure 6) can be $ValueIteration$ or $PolicyIteration$. We used the latter during the competition since it seems to be more original and sometimes more efficient than the former. *SFDP Policy Iteration* is implemented by the algorithm shown in Figure 8: the generic function $Reachable$ is applied from the initial state set $I$ at each iteration, using actions from the current policy $\Pi_k$, until $G$ is reached. As a matter of fact, the working space $S_k$ of *SFDP* is absolutely not guaranteed to grow : on the contrary, *SFDP* has been designed to *focus* on coherent parts of the state space as shown in Figure 7. As a consequence, *SFDP* will not give the optimal solution to the problem, rather the "shortest solution", unless *SFDP* is compelled to visit all the rewards of the state space because all the rewards of the problem have been given as *planning goals constraints* prior to the optimization. For *sLAO**, we implemented the algorithm shown in Figure 9, where the generic function $Reachable$ is applied from $S_k$ at each iteration, using actions from the current policy $\Pi_k$, with $1\ step\ lookahead$. Contrary to *SFDP*, $S_k$ in *sLAO** is always supposed to grow, which in that context gives *sLAO** a guarantee of optimality.

**Init**
$R_0 \leftarrow Reachable(I, \mathcal{A}, G)$
$\Pi_0 \leftarrow ShortestStochasticPath(R_0 \rightarrow G)$
$S_0 \leftarrow FilterStates(R_0, P(s) < \epsilon \cdot P(I))$
$\quad \Longrightarrow (\Pi_0, S_0)$

$k \leftarrow 0$
**repeat**
$\quad S_{k+1} \leftarrow Reachable(I, \Pi_k, G)$
$\quad PolicyIteration(\Pi_k, S_{k+1})$
$\quad k \leftarrow k + 1$
**until** convergence over $S_k$

Figure 8: *SFDP Policy Iteration*

**Init**
$R_0 \leftarrow Reachable(I, \mathcal{A}, G)$
$\Pi_0 \leftarrow ShortestStochasticPath(R_0 \rightarrow G)$
$S_0 \leftarrow FilterStates(R_0, P(s) < \epsilon \cdot P(I))$
$\quad \Longrightarrow (\Pi_0, S_0)$

$k \leftarrow 0$
**repeat**
$\quad S_{k+1} \leftarrow Reachable(S_k, \Pi_k,\ 1\ step\ lookahead)$
$\quad PolicyIteration(\Pi_k, S_{k+1})$
$\quad k \leftarrow k + 1$
**until** convergence over $S_k$

Figure 9: *sLAO* Policy Iteration*

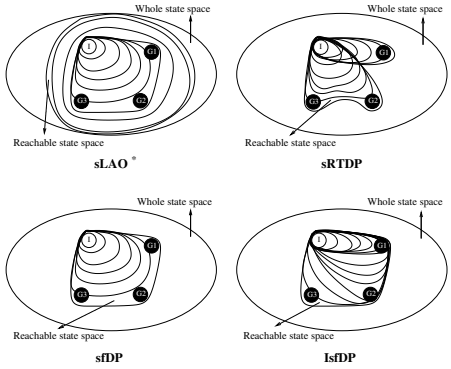



Figure 10: Reachable state spaces of sLAO*, sRTDP, sfDP and IsfDP

## Experimentations

We conducted our experiments on gridworld exploration problems inspired from the example shown in figure 1. The number of nodes of the navigation graph is $45 \times 45 = 2025$ and the total number of states grows exponentially in the number of additional state variables: goals, regions in the problem (+1 for the energy level).

We first performed the comparison between the factored MDP and enumerated MDP approaches. In linear problems, each region is only linked with a previous and a next region. In concentric problems, each region is linked to a central region. Results are presented in Figure 11 that shows that state space modeling is really a crucial issue. Columns are for problems type (either "linear" or "concentric"), state space $size$, number $rg.$ of regions, model of the MDP ($F$ for factored MDP and $E$ for enumerated MDP), time $B.$ for building the MDP, time $D.$ for decomposition, number $\pi_r$ of macro-actions (local policies computed), total solution time $T.$. The time required for the building of the state transition data structures for the enumerated MDP illustrates handicap

of the enumerated approach. Problems of larger size could not be solved in comparable time: this is why they do not appear in the table. The complexity burden is apparently higher for "concentric problems" than for "linear problems" (compare the complexity step between the "linear problems" and the "concentric problems").

| type | size | rg. | model | B. | D. | $\pi_r$ | T. |
|---|---|---|---|---|---|---|---|
| linear | 384 | 3 | F | $< 0.01$ | 0.08 | 10 | 0.02 |
| | | | E | 0.03 | – | – | 0.01 |
| | $6 \cdot 10^3$ | 6 | F | 0.01 | 0.38 | 33 | 1.51 |
| | | | E | 4.21 | – | – | 0.38 |
| | $7 \cdot 10^4$ | 9 | F | 0.03 | 0.56 | 47 | 26.8 |
| | | | E | 587.62 | – | – | 5.03 |
| concentric | $8 \cdot 10^5$ | 9 (9 s./r.) | F | 0.02 | 0.13 | 21 | 0.12 |
| | | | E | 746.98 | – | – | 2.25 |
| | $7 \cdot 10^6$ | 9 (81 s./r.) | F | 0.02 | 40.61 | 61 | 16.77 |
| | | | E | $>$ 1hr | – | – | – |

Figure 11: Comparison between the factored MDP and enumerated MDP approaches (elapsed time in seconds)

We then present a comparison of six algorithms that have been implemented on the basis of the SPUDD/VI *value iteration* algorithm: 1.SPUDD/VI – 2.SPUDD/PI – 3.sLAO/VI – 4.sLAO/PI – 5.SFDP/VI – 6.SFDP/PI. Note that the algorithm number 4 participated in the ICAPS'04 competition but it was not as mature as today. We present results obtained with stochastic exploration-like problems because they are closer to our research projects than the competition problems. Yet, the complexity of such exploration problems is comparable with the ICAPS'04 probabilistic track competition problems. We have compared the solution quality

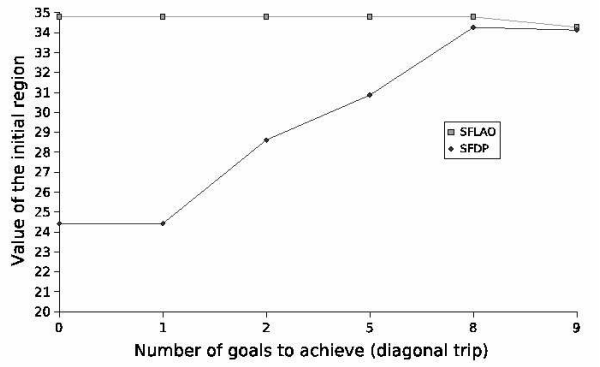



Figure 12: *SFDP* Solution quality (Value at starting point) compared with *sLAO** while increasing the number of planning goals to achieve

of *SFDP* policies when a growing number of constraints are imposed on both algorithms concerning the planning goals. It appears that *SFDP* appears as much more sensitive to goal constraints than *sLAO**. Imposing on *SFDP* to achieve ALL the goals leads the algorithm to behave like *sLAO**, continuously extending its planning space without *focusing*, as long as the corresponding problems remains tractable. On the contrary, *sLAO** tends to try and reach all the rewards and goals of the problem even when it is not asked so. The corresponding computation time grows in proportions with the number of combinations of alternatives.

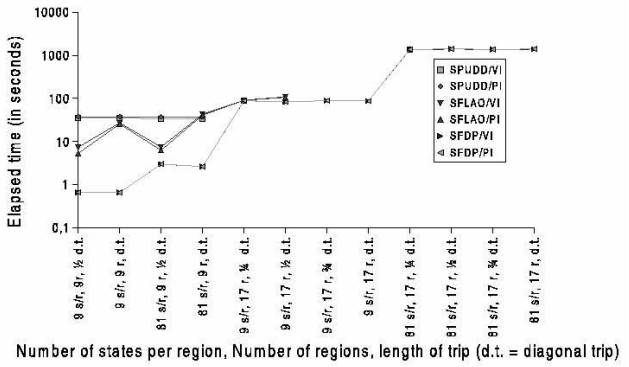



Figure 13: *SFDP* Solution time compared with *sLAO** and *SPUDD* for different starting points

We have similarly compared the computation time for *sLAO** and *SFDP* on problems of growing complexity (varying the starting and goal points). The conclusion is that *SFDP*, still finds quite quickly (sub-optimal) solutions for the most complex problems that we could decompose in reasonable time (248s) and without swapping, which corresponds to the diagonal trip. By contrast, *sLAO** cannot give any answer after more than one hour on the fourth problem. Such comparison should be analyzed carefully: *SFDP* cannot be considered as "better" nor "preferable" on the basis of this comparison. On the other hand, Figure 12 shows that it is possible to establish a quality response profile for *SFDP* on some classes of problems. The quick answer given by this algorithm could be reused in order to initiate an admissible heuristic policy, or value function for another algorithm.

## Faster solution, weaker value

Following the previous ideas, another version of the *focused dynamic programming* scheme was developed by weakening the stopping condition of the $Reachable$ function. The new stopping condition holds as soon as **at least one state is reached where the required goal conditions are achieved**. This new stopping condition is obviously weaker and the new algorithms, called *sfDP* and *sfLAO* still follow the schemes respectively presented in Figure 8 and 9, but with the new $Reachable$ function. The *sfDP* and *sfLAO* algorithms are obviously not optimal, but show interesting properties in terms of incremental behavior, as shown in Figures 14 and 15. Experimental results presented in Figure 16 show that *sfLAO* finds better solution than *sfDP*, with similar computation times. Interestingly enough, *sfLAO* still shows the same behavior exhibited with *SFDP*: the solution quality grows with the number of goal conditions imposed up to the optimal solution. As a consequence, the computation time required in order to obtain the optimal solution is also a growing function and reaches the elapsed time obtained with *sLAO**, as shown in Figure 14.

As a matter of fact, we used this idea to develop an incremental version of both *sfDP* and *sfLAO* algorithms, with respectively the *IsfDP* and *IsfLAO* algorithms that are described in Figures 17 and 10. Experimentations shown in

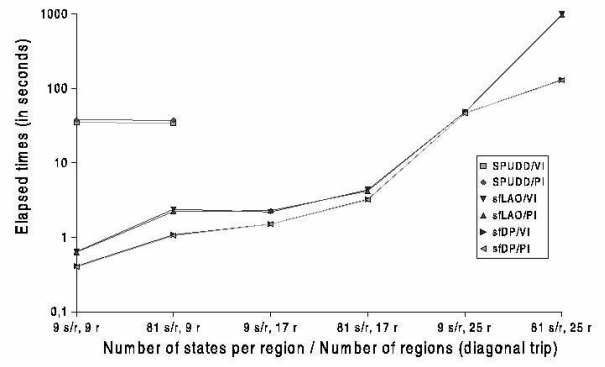



Figure 14: sfDP and sfLAO optimization time while increasing the problem size

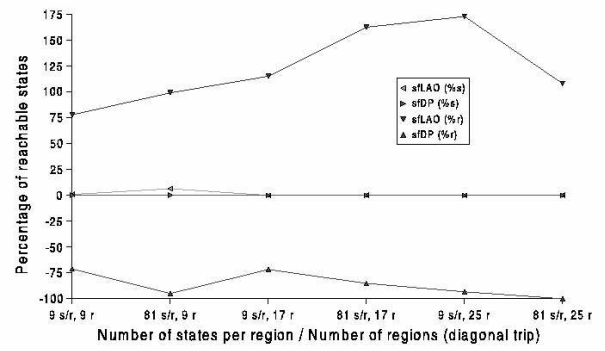



Figure 15: Percentage ($\%s$) of explored reachable states and variation ($\%r$) for sfDP and sfLAO

Figures 18 and 19 show that *IsfDP* clearly outperforms *IsfLAO*. In conclusion, *sfLAO* algorithms is interesting for improving the quality of the solution and could be used in a last iteration on the basis of previous optimization performed by *IsfDP*. An optimal scheme could be designed on this basis.

## Related Work

The heuristic search dynamic programming schemes discussed in this paper have common features with for example recent work on *sRTDP* (Feng, Hansen, & Zilberstein 2003), a *symbolic* on-line version of *RTDP*. However, the *SFDP* algorithm and the way its behavior can be controlled through planning goals constraints is a fully original contribution to our knowledge. Previous comparisons (Teichteil & Fabiani 2005) between *LAO*-like* algorithms and *RTDP-like* algorithms have shown that *LAO** would be better when the problem topology is "open" and *RTDP-like* algorithms would be more efficient in "corridors"., as it seems to be the case in (Bonet & Geffner 2003b). Both heuristic schemes lead to limit the size of the explored state space before convergence. Their respective application in (Feng & Hansen 2002) and (Aberdeen, Thibaux, & Zhang 2004) show that they improve the efficiency of value iteration dynamic programming for structured MDPs. Different implementations of these heuristic search value iteration algorithms were independently compared on navigation grid MDPs in (Hansen & Shlomo 2001) and in (Bonet & Geffner 2003b), where LRTDP outperforms LAO*. On symbolic stochastic problems, results in (Feng & Hansen 2002) and (Feng, Hansen, & Zilberstein 2003) show that *sRTDP* presents a faster on-line performance while sLAO* shows a better off-line convergence efficiency. Furthermore, (Bonet & Geffner 2003a) propose a general heuristic dynamic programming scheme *FIND-and-REVISE* that is different but might be confused with our two-phased scheme: *planning space expansion*-and-*dynamic programming*.

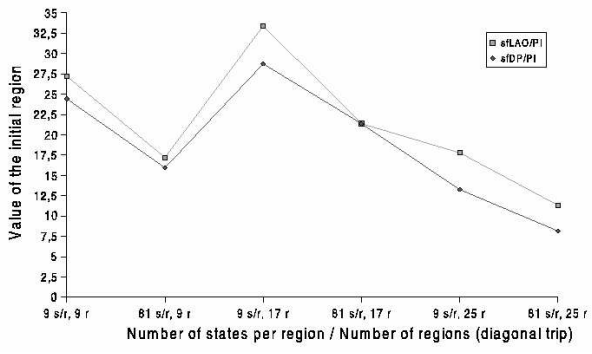



Figure 16: sfDP and sfLAO Solution Quality (Value at starting point) while increasing the problem size

$L_{sg} \longleftarrow$ List of subgoals to achieve
$S_0 \longleftarrow I$
$n \longleftarrow 1$
**while** $L_{sg}$ non empty **do**
  $I_n \longleftarrow S_{n-1}$
  $G_n \longleftarrow$ head of $L_{sg}$
  $S_n \longleftarrow SFDP(I_n, G_n)$
  remove head of $L_{sg}$
  $n \longleftarrow n+1$
**end while**

Figure 17: IsfDP algorithm for on-line planning

## Conclusion

We have proposed an original algorithm *SFDP* for which the optimization time and the solution quality can be controlled through the definition of planning goals constraints. The design of *SFDP*, and the principles of the proposed underlying *focused dynamic programming* scheme, meet the challenges of planning under uncertainty in large state spaces for autonomous systems that rather need a current solution quite rapidly, and an optimal one if possible. Goal conditions can be adapted off-line or on-line, thus opening interesting directions for future work on decision under time and ressources constraints. This is particularly interesting in our application perspectives on autonomous aircraft. Among possible perspectives, we will consider carefully deriving an on-line version *OSFDP* that would adjust on-line the number of goal constraints to satisfy in response to the available time for finding a solution to the problem. We will also consider

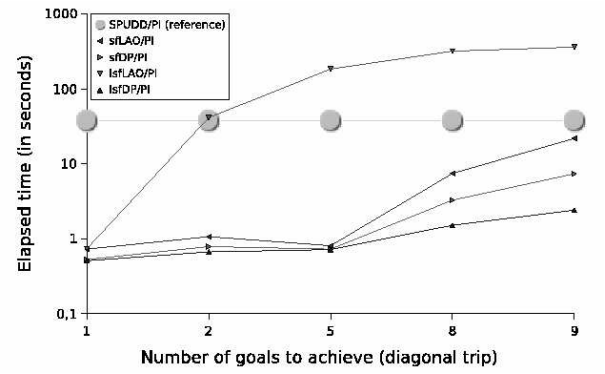



Figure 18: sfDP, IsfDP, sfLAO and IsfLAO optimization time compared with SPUDD while increasing the number of planning goals to achieve

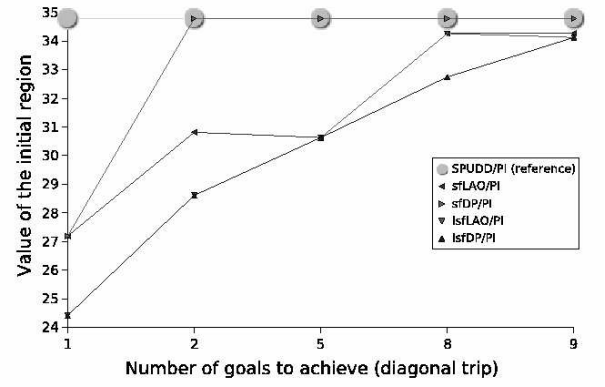



Figure 19: sfDP, IsfDP, sfLAO and IsfLAO Solution Quality (Value at starting point) compared with SPUDD while increasing the number of planning goals to achieve

the coupling of *SFDP* with a higher level optimization controller in order to reuse the sub-optimal solution obtained with *SFDP* in a higher level optimization process. This was shown in the development of the incremental *IsfDP* algorithm, based on *sfDP*, an even faster, but weaker, version *sfDP* of the *focused dynamic programming* scheme. *sfDP* presented that quickly finds solution in larger problems. The incremental version *IsfDP* incrementally improves the current solution thanks to iterative calls of *sfDP* with an increasing list of planning subgoals to be taken into account. We have compared all these algorithms on a set of problems of different sizes. We will now develop an optimal version of the *focused dynamic programming* scheme, that would provide rapidly a current solution even on large state space problems, but would improve it up to the optimal solution as time is available.

## References


Aberdeen, D.; Thibaux, S.; and Zhang, L. 2004. Decision-theoretic military operations planning. In *Proceedings 14th ICAPS 2004*, 402–412.

Bacchus, F.; Boutilier, C.; and Grove, A. 1997. Structured solution methods for non-markovian decision processes. In *Proceedings 14th AAAI*, 112–117.

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

Bonet, B., and Geffner, H. 2003a. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI*, 1233–1238.

Bonet, B., and Geffner, H. 2003b. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *Proceedings 13th ICAPS 2003*, 12–21.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *J.A.I.R.* 11:1–94.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1-2):49–107.

Cassandra, A. R. 1998. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. Computer science, U. of Illinois, Providence R.I.

Dean, T., and Kanazawa, K. 1989a model for reasoning about persistence and causation*Computational Intelligence* 5(3): 142–150.

Dean, T., and Lin, S.-H. 1995. Decomposition techniques for planning in stochastic domains. In *Proceedings 14th IJCAI*, 1121–1129.

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89:219–283.

Feng, Z., and Hansen, E. 2002. Symbolic heuristic search for factored markov decision processes. In *Proceedings 18th AAAI*, 455–460.

Feng, Z.; Hansen, E. A.; and Zilberstein, S. 2003. Symbolic generalization for on-line planning. In *Proceedings 19th UAI*, 209–216.

Hansen, E. A., and Shlomo, Z. 2001. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129:35–62.

Hauskrecht, M.; Meuleau, N.; Kaelbling, L. P.; Dean, T. L.; and Boutilier, C. 1998. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings 14th UAI*, 220–229.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 2000. Optimal and approximate stochastic planning using decision diagrams. Technical Report TR-2000-05, University of British Columbia.

Parr, R. 1998. Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings 14th UAI*, 422–430.

Puterman, M. L. 1994. *Markov Decision Processes*. John Wiley & Sons, INC.

Teichteil, F., and Fabiani, P. 2005. Influence of modeling structure in probabilistic sequential decision problems. *RAIRO Operations Research* to appear.

# Distributed re-planning with bounded time resources in the ARTEMIS project

Patrick Fabiani
Eric Bensana
Jean-Loup Farges
ONERA-CERT / DCSD / CD
BP 4025
31055 TOULOUSE, France
fabiani@cert.fr, bensana@cert.fr,
farges@cert.fr

Philippe Morignot
Jean-Clair Poncet
Johan Baltié
AXLOG Ingéniérie
19-21, rue du 8 mai 1945
ARCUEIL, France
philippe.morignot@axlog.fr,
jean-clair.poncet@axlog.fr,
johan.baltie@axlog.fr

Bruno Patin
Dassault Aviation
78, quai Marcel Dassault
92552 SAINT-CLOUD
Cedex, France
bruno.patin@dassault-aviation.fr

**Abstract**

This paper shortly presents some aspects of the ARTEMIS project dedicated to the development and testing of algorithms for distributed planning with bounded time resources. The ARTEMIS project addresses the problem of future mission management systems (MMS) for uninhabited air combat vehicles (UCAVs) acting in cooperation (possibly with other inhabited aircraft) in combat mission such as the attack of an enemy target. The challenges for such MMS are related to both the architectures and the algorithms for autonomous decision and information processing onboard such UCAVs. We present the distributed re-planning problem under time constraints and a sketch of our approach to solve it.

## INTRODUCTION

Significant research efforts have been devoted to new technologies needed for the development of onboard mission management systems for uninhabited air vehicles (UAVs). As a matter of fact, current UAV systems are mostly remotely controlled by an operator that can control the aircraft flight plan by choosing way-points or targets to be achieved by an on-board auto-pilot. Current concepts of use of UAVs are very close to the use of aircraft in the early years of aviation: missions of reconnaissance, observation, etc. These concepts of use are very likely to change with progress in the area of UAV autonomy.

Constraint model-based planning is a promising framework for such applications [1]. The MISURE project [2] addresses the feasibility of a mission management system using similar constraint programming techniques for a package of UCAVs involved in an attack mission. However, limited computation time resources and distribution constraints are not addressed in MISURE.

The ARTEMIS project [3] takes into account these two constraints. We develop algorithms for distributed re-planning with bounded computation time resources. We also consider the possible intervention of the operator, but rather as an external event that may cause the MMS to re-plan the mission. Our efforts are more focused on the necessity of bounded time cooperation schemes among distributed planning agents in order to achieve coordination and collaboration towards common goals while dealing with communication limitations. The agent decisional architecture, inspired from autonomous robots, was prototyped using ProCoSA [4].

## MISSION

The mission considered in the ARTEMIS project roughly consist in the attack by a package of 4 to 8 UCAVs of a number of targets in enemy zone. The UCAVs can organize their cooperation at planning time. The package can split in sub-groups of UCAVs that can realize sub-tasks before rejoining other Due to communication limitations

Figure 1 shows a sketch of the problem situation. Before the mission starts, a mission pre-planning phase always occur, during which :

- primary (and secondary) targets are designated,
- known threats are identified,
- the airspace is divided into a number of zones that expresses tactical constraints such as : the "attack zone", "flight corridors", etc. Corridors are 3D volumes associated with time constraints.

## CONSTRAINT BASED RE-PLANNING

The constraint model-based planning problem is defined in a form similar to the work in [1] and [2], but in a distributed re-planning context. The implementation of each agent re-planner is made using the CHOCO [5] library. A mixed integer programming version of the re-planning problem using CPLEX [6] was also developed in order to have a reference solution on simple instances, so as to study the possible loss of performance caused by the computation time and distribution constraints in the solution of the re-planning problem. Each agent re-plans in bounded time by applying the decision-repair paradigm proposed in [7], combining a local-search scheme (such as Tabu search [8]), the management of explanations and constraint propagation techniques.

## COOPERATION BETWEEN PLANNERS

The cooperation between distributed agents is based on the coordination of the distributed planners within a communication group: a group of UCAVs that can communicate with each other without limitations during the bounded time re-planning process.

Different cooperation schemes have been envisioned for the ARTEMIS project. Few of them can be efficiently controlled in order to guarantee a bounded time response in all circumstances. The scheme that we selected is based on a **selection of propositions** [9] [10] by a planning coordinator within a communication group. This type of method is not iterative and involves basically two steps. The group coordinator initiates the re-planning process. Each agent formulates a finite set of propositions about the actions it could perform. The coordinator selects a single proposition per agent such that interaction constraints are satisfied and optimizes a global criterion, at the group level. Further refinement are possible if more time is available.

## CONCLUSION

We have presented the ARTEMIS project, its current status and the ongoing developments. Uncertainties are partly taken into account, for instance via some probabilities of kill or of survival. Beyond ARTEMIS, a more thorough combination of constraint based planning and a planning under uncertainty framework should be studied in order to better address real world problems.

## ACKNOWLEDGEMENT

This work is funded by the Délégation Générale pour l'Armement (from the French MoD) via the contract ARTEMIS. We thank Nelly Strady-Lécubin, Catherine Tessier, Jean-François Gabard, Stéphane Millet and Pierre Hélie.

## REFERENCES

[1] C. Guettier, B. Allo, V. Legendre, J.-C., Poncet, N. Strady-Lécubin. Constraint Model-Based Planning and Scheduling with Multiple Ressources and Complex Collaboration Schema. In *Proceedings of the Sixth International Conference on A.I. Planning and Scheduling*. Toulouse, France, April 2002, pages 183-194.

[2] J.-C. Poncet. Mission Management System for Uninhabited aiR vEhicles (MISURE), *Technical Note 3.1 Identification of relevant software techniques for approaching the mission management and formation problems*, Eurofinder Project, 2004.

[3] P. Morignot, P. Fabiani, J.-F. Gabard, B. Patin, S. Millet. *Simulating Uninhabited Combat Aircraft in Hostile Environments*. In *Proceedings of the Spring Simulation Interoperability Workshop*, SIW SISO'04, Arlington, VA, April 2004, 9 pages, ref. 04S-SIW-081.

[4] ProCoSA supervision software available at URL http://www.cert.fr/dcsd/cd/PROCOSA

[5] F. Laburthe. CHOCO. Available at URL http://choco.sourceforge.net

[6] I. Lustig. CPLEX Reference Manual, ILOG, Gentilly, France, 2004.

[7] N. Jussien, O. Lhomme. Local Search with Constraint Propagation and Conflict-based Heuristic. *Artificial Intelligence*, vol. 139, p21-45, 2002.

[9] F. Glover, M. Laguna. *Tabu Search*. Kluwer, Boston, 1997.

[9] R.W. Beard, T.W. McLain, M. Goodrich, E.P. Anderson. Coordinated Target Assignment and Intercept for Unmanned Air Vehicles. *IEEE Transactions on Robotics and Automation*, Vol. 18, n° 6, p911-922, December 2002.

[10] Y. Kuwata, *Real-time Trajectory Design for Unmanned Aerial Vehicles using Receding Horizon Control.* MIT MSc thesis, 2003.

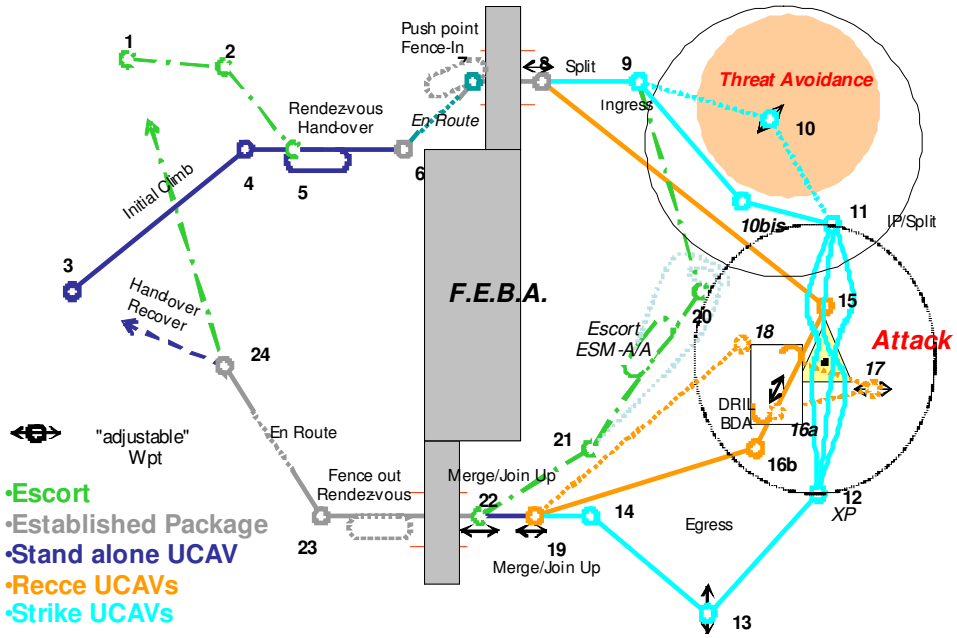



**Figure 1** : mission for a package of UCAVs.

# The Autonomous On-Board Mission Planning System for BayernSat

**Robert Axmann, Martin Wickler**

- Department for Planning and Optimization Technology -
German Space Operations Center (GSOC)
Oberpfaffenhofen, 82234 Wessling, Germany

Robert.Axmann@dlr.de
Martin.Wickler@dlr.de

**Abstract**

A future trend in the operation of spacecraft and other space vehicles is the increasing system autonomy, also in the area of mission planning. Increased autonomy leads to reduced reaction times and more optimized usage of spacecraft resources. It can also reduce the amount of human interaction in ground based space operation centers.
The German Space Operations Center (GSOC) developed and used its own ground based mission planning system since several years. In a next step a system will be developed to allow mission planning on-board of a spacecraft. The Autonomous On-board Mission Planning System (AOMPS) should demonstrate the possibility to autonomously plan activities and react in changing environments during the mission (real-time replanning). First usage is planned for BayernSat, a technology demonstration satellite developed by Technische Universität München.

## Development Background

Spacecraft operations are increasingly autonomous operations. On-board autonomy in terms of mission planning is mainly driven by the available on-board data processing capabilities. Current spacecraft systems are equipped with processor power in the range of up to approx. 100 MHz and several tens of MB of RAM. Mission planning needs exponentially growing computing power with each added degree of freedom. The availability and usage of better hardware components for spacecraft on-board data processing is therefore mandatory to allow for higher processing capabilities in orbit, a basic prerequisite for the usage of a mission planning software like AOMPS.
BayernSat (developed and built by the Technische Universität München) as a technology demonstration satellite offers the possibility to develop and test an autonomous on-board mission planning system. A PowerPC will be used on-board with planned 800 MHz (1800 MIPS) processing power and approx. 256 MB of RAM. The satellites orbit is a low earth orbit without any on-board propulsion. During its lifetime the telepresence technique is demonstrated by sending live video streams down to earth. A portion of the available PowerPC computing power is not used by the satellite and a therefore available to the AOMPS.
Development of AOMPS will be based on the existing GSOC mission planning tools. The software has been developed since the 90's. Tool development was driven and accompanied by the feedback from operational usage during different missions. MIR97, GRACE and SRTM are examples for successful prepared and planned missions.
The mission planning system consists mainly of different tools for activity modeling (PINTA), activity scheduling (PLATO) and visual publication of the planned activities (TimOnWeb). Mission planning with these different parts of software has been done on ground during mission preparation and during the mission itself. Common to all these missions is the relatively high involvement of ground operations staff for planning. Autonomous working on-board software can reduce the needed staff, decrease reaction time and increase spacecraft resource utilization.

## Autonomous On-Board Mission Planning System Development

With the background of higher processing capabilities on BayernSat and the experience with the actual GSOC mission planning system the aim is to develop as far as possible an autonomous on-board mission planning system. Because the satellite uses RTEMS as the operating system (OS) for the PowerPC boards the software will also be developed for this OS. C or C++ will be the used programming language which should allow for easily porting the software to other OS. Development for a real time operating system as RTEMS is should also reduce the programming overhead to a minimum. Based on past

experience not more than 5 MB of on-board satellite memory should be needed for AOMPS.
The planning process of the software could be split into system planning and payload planning. An example for a system planning activity is charging of batteries or the desaturation of reaction wheels. Payload related activities focus on the operation of the spacecrafts payloads. This may include heating or cooling of an instrument and positioning of the satellite in advance to the imaging opportunity. Distinction between system and payload related activities are not each time possible as shown in the last example. It includes also system activities as changing attitude by usage of thrusters or reaction wheels.
The focus of the BayernSat on-board planning software is primary on payload activity planning. This focus has been selected to allow for a staggered approach due to security issues of the satellite.
In difference to conventional planning systems the on-board planning system should be capable to react on planning relevant events in real time within some seconds after occurrence of the triggering event.
As an example, a data take should be replanned automatically if the instrument sends an error status or the satellite batteries are discharged and not able to support a payload operation. Another problem often experienced in earth observation missions with optical sensors is the problem of cloud coverage. This event can also trigger the real time replanning process of the system.

To realize mission planning in advance and intelligent real-time reaction on changing environmental (mission planning) conditions its necessary to have the planning software integrated in the other on-board software packages. In principle this means to get all the required system and payload status data. On the other site it's necessary to have an interface to command the satellite in order to execute the planned activities.

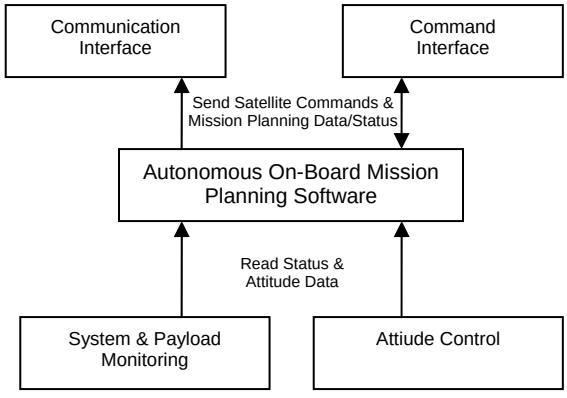



**Figure 1: Principle integration of an on-board mission planning system**

Figure 1 depicts the principle integration approach used for integration of AOMPS into the on-board software of BayernSat. The software is integrated into the other software additionally to make conventional operation modes possible. AOMPS receives monitoring data for payload and system. Attitude data is received from the attitude control software. All the information is used by the software to determine the health status of the satellite and derive adequate changes in activity planning.

Besides receiving status information from satellite it also necessary to send commands to the satellite. As for normal ground based commanding a layer of abstraction is used for security reasons. Commands from ground are received by the command interface and processed. This interfaces checks validity of commands and executes them only if possible. The autonomous mission planning agent also uses this interface. This reduces the risk of causing satellite bus and payload failures by directly sending commands to the hardware. Also command execution information is received via this interface.

All data exchange with the ground is done with help of the communication interface. The mission planning software uses this interface for downlink of satellite status information and results of the activity execution (e.g. images or other payload data). Also information about the mission planning software itself is send down to ground. This allows for analysis of the software function. Operation of AOMPS is optional and can be switched on and off during mission.

## Conclusion

System development has been started beginning 2005 together with the satellite project BayernSat. The satellite offers a unique capability due to its technology demonstration approach and computer performance. Nevertheless reliability and on-board interface definition needs special attention during software development. Therefore a staggered approach has been chosen for AOMPS, allowing the software to be operated optionally and with different stages of autonomy. Software will be developed in the C or C++ language for RTEMS as the satellites operating system. It is assumed that the software code for AOMPS needs less than 5 MB of the satellites on-board memory.

## References

Department for Planning and Optimization Technology: *www.dlr.de/rb/institut/gsoc/planningtechnology*

Project BayernSat:
*www.bayernsat.com*
*www.lrt.mw.tum.de*