

A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains

Nicolas Meuleau

NASA Ames Research Center
Mail Stop 269-3
Moffet Field, CA 94035-1000, USA

NMEULEAU@EMAIL.ARC.NASA.GOV*

Emmanuel Benazera

Universität Bremen
Fachbereich 3 - AG Robotik
Robert-Hooke- Str. 5
D-28359 Bremen, Germany

BENAZERA@INFORMATIK.UNI-BREMEN.DE

Ronen I. Brafman

Department of Computer Science
Ben-Gurion University
Beer-Sheva 84105, Israel

BRAFMAN@CS.BGU.AC.IL

Eric A. Hansen

Dept. of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762, USA

HANSEN@CSE.MSSTATE.EDU

Peter Lamborn

Dept. of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762, USA

PCL16@MSSTATE.EDU

Mausam

Dept. of Computer Science and Engineering
University of Washington
Seattle, WA 981952350, USA

MAUSAM@CS.WASHINGTON.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 depends on resource availability. 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. Like other heuristic search algorithms, HAO* leverages knowledge of the starting state and an admissible heuristic to focus computational effort on those parts of the state space that could be reached from the start state by following an optimal policy. We show that this approach is especially effective when resource constraints limit how much of the state space is reachable. Experimental results demonstrate its effectiveness in the domain that motivates our research – automated planning for planetary exploration rovers.

1. Introduction

Many NASA planetary exploration missions rely on rovers – mobile robots that carry a suite of scientific instruments for use in characterizing planetary surfaces and transmitting information back to Earth. Because of difficulties in communicating with devices on distant planets, direct human control of rovers by teleoperation is infeasible, and rovers must be able to act autonomously for substantial periods of time. For

*. Carnegie Mellon University

example, the Mars Exploration Rovers (MER) are designed to communicate with the ground only twice per Martian day.

Autonomous control of planetary exploration rovers presents many challenges for research in automated planning. Progress has been made in meeting some of these challenges. For example, the planning software developed for the Mars Sojourner and MER rovers has contributed significantly to the success of these missions (Bresina, Jonsson, Morris, & Rajan, 2005). But many important challenges must still be addressed to achieve the more ambitious goals of future missions (Bresina, Dearden, Meuleau, Ramakrishnan, Smith, & Washington, 2002).

Among these challenges is the problem of plan execution in uncertain environments. On planetary surfaces such as Mars, there is uncertainty about the terrain, meteorological conditions, and the state of the rover itself (position, battery charge, solar panels, component wear, etc.) In turn, this leads to uncertainty about the outcome of the rover's actions. Much of this uncertainty is about resource consumption. For example, factors such as slope and terrain affect speed of movement and rate of power consumption, making it difficult to predict with certainty how long it will take for a rover to travel between two points, or how much power it will consume in doing so. Because of limits on critical resources such as time and battery power, rover plans are currently very conservative and based on worst-case estimates of time and resource usage. In addition, instructions sent to planetary rovers are currently in the form of a sequential plan for attaining a single goal (e.g., photographing an interesting rock). If an action has an unintended outcome that causes a plan to fail, the rover stops and waits for further instructions; it makes no attempt to recover or achieve an alternative goal. This can result in under-utilized resources and missed science opportunities.

Over the past decade, there has been a great deal of research on how to generate conditional plans in domains with uncertain action outcomes. Much of this work is formalized in the framework of Markov decision processes (Puterman, 1994; Boutilier, Dean, & Hanks, 1999). However, as Bresina et al. (2002) point out, important aspects of the rover planning problem are not adequately handled by traditional planning algorithms, including algorithms for Markov decision processes. In particular, most traditional planners assume a discrete state space and a small discrete number of action outcomes. But in automated planning for planetary exploration rovers, critical resources such as time and battery power are continuous, and most of the uncertainty in the domain results from the effect of actions on these variables. This requires a conditional planner that can branch not only on discrete action outcomes, but on the availability of continuous resources, and such a planner must be able to reason about continuous as well as discrete state variables.

Closely related to the challenges of uncertain plan execution and continuous resources is the challenge of over-subscription planning. The rovers of future missions will have much improved capabilities. Whereas the current MER rovers require an average of three days to visit a single rock, progress in areas such as automatic instrument placement will allow rovers to visit multiple rocks and perform a large number of scientific observations in a single communication cycle (Pedersen, Smith, Deans, Sargent, Kunz, Lees, & Rajagopalan, 2005). Moreover, communication cycles will lengthen substantially in more distant missions to the moons of Jupiter and Saturn, requiring longer periods of autonomous behavior. As a result, space scientists of future missions are expected to specify a large number of science goals at once, and often this will present an *over-subscription planning problem*, that is, a problem in which it is infeasible to achieve all goals, and the objective is to achieve the best subset of goals within resource constraints (Smith, 2004). In the case of the rover, there will be multiple locations the rover could reach, and many experiments the rover could conduct, most combinations of which are infeasible due to resource constraints. The planner must select a feasible subset of these that maximizes expected science return. When action outcomes (including resource consumption) are stochastic, a plan that maximizes expected science return will be a conditional plan that prescribes different courses of action based on the results of previous actions, including resource availability.

In this paper, we present an implemented planning algorithm that handles all of these problems together: uncertain action outcomes, limited continuous resources, and oversubscription planning. We formalize the rover planning problem as a hybrid Markov decision processes, that is, a Markov decision process with both discrete and continuous state variables, where the continuous variables are used to represent limited resources. The planning algorithm we introduce builds on earlier work on dynamic programming algorithms for hybrid MDPs, in particular, the work of Feng *et al.* (2004). However, our algorithm is not a dynamic programming

algorithm, but a heuristic search algorithm. Called HAO*, for Hybrid AO*, it is a generalization of the classic AO* heuristic search algorithm (Nilsson, 1980; Pearl, 1984). Whereas AO* searches in discrete state spaces, HAO* solves planning problems in hybrid domains with both discrete and continuous state variables, where the continuous variables represent limited resources, and it constructs conditional plans that branch on the values of both the discrete and continuous state variables.

It is well-known that heuristic search can be more efficient than dynamic programming because it uses reachability analysis guided by a heuristic to focus computation on the relevant parts of the state space. We show that for problems with resource constraints, including over-subscription planning problems, this advantage is greater than usual because resource constraints can significantly limit reachability. Unlike dynamic programming, forward search keeps track of the trajectory from the start state to each reachable state, and thus 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, due to resource constraints. Although our work is motivated by the rover planning problem, we develop a general planning algorithm that could be used to solve many other problems. For example, it could be used to solve a large class of logistics problems with uncertain (stochastic) travel times and deadlines.

The paper is structured as follows. Section 2 formalizes the rover planning problem as a hybrid Markov decision process and reviews dynamic programming algorithms for solving hybrid MDPs. Section 3 introduces the HAO* algorithm for searching in hybrid state spaces and analyzes its properties. Section 4 describes the results of experiments conducted with HAO* on a realistic model of a NASA planetary exploration rover. We discuss future work and conclude in Section 5.

2. Problem formulation and background

We start with a formal definition of the planning problem we are tackling. Since it is a special case of a hybrid Markov decision process, we begin by defining this model. Then we discuss how to include resource constraints and formalize over-subscription planning in this model. Finally we review a class of dynamic programming algorithms for solving hybrid MDPs, since some of these algorithmic techniques will be incorporated in the heuristic search algorithm we develop in Section 3.

2.1 Hybrid Markov decision process

A hybrid Markov decision process, or *hybrid MDP*, is a factored Markov decision process that has both discrete and continuous state variables. We define it as a tuple (N, \mathbf{X}, A, P, R) , where N is a discrete state variable, $\mathbf{X} = \{X_1, X_2, \dots, X_d\}$ is a set of continuous state variables, A is a set of actions, P is a stochastic state transition model, and R is a reward function. We next describe these elements in more detail.

Although a hybrid MDP can have multiple discrete variables, this plays no role in the algorithms described in this paper, and so, for notational convenience, we model the discrete component of the state space as a single variable N . In most application domains, the discrete state has a factored representation. The variable N is then a flat representation of an otherwise structured discrete state. Our focus is on the continuous component. We assume the domain of each continuous variable $X_i \in \mathbf{X}$ is a closed interval of the real line, and so $\mathbf{X} = \otimes_i X_i$ is the hypercube over which the continuous variables are defined. The state set S of a hybrid MDP is the set of all possible assignments of values to the state variables. In particular, a *hybrid state* $s \in S$ is a pair (n, \mathbf{x}) where $n \in N$ is the value of the discrete variable, and $\mathbf{x} = (x_i)$ is a vector of values of the continuous variables.

State transitions occur as a result of *actions*, and the process evolves according to Markovian *state transition probabilities* $\Pr(s' | s, a)$, 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. These probabilities can be 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') , $\int_{\mathbf{x}' \in \mathbf{X}} \Pr(\mathbf{x}'|n, \mathbf{x}, a, n') d\mathbf{x}' = 1$.

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

2.2 Resource constraints and over-subscription planning

To model the rover planning problem, we consider a special type of MDP in which the objective is to optimize expected cumulative reward *subject to resource constraints*. We make the following assumptions:

- there is an initial allocation of one or more non-replenishable resources,
- each action has some minimum positive consumption of at least one resource,
- and once resources are exhausted, no further action can be taken.

One way to model this is as a *constrained MDP*, a model that has been widely studied in the operations research community (Altman, 1999). In this model, each action a incurs a transition-dependent resource cost, $C_a^i(s, s')$, for each resource i . Given an initial allocation of resources and an initial state, linear programming is used to find the best feasible policy, which may be a randomized policy. Although a constrained MDP models resource consumption, it does not include resources in the state space. As a result, a policy cannot be conditioned upon resource availability. This is not a problem if resource consumption is deterministic or if it is unobservable. But it is not a good fit for the rover domain, in which resource consumption is stochastic, and the rover may take different actions depending on current resource availability.

We adopt a different approach to modeling resource constraints in which resources are included in the state description. Although this increases the size of the state space, it allows decisions to be made based on resource availability, and it allows a stochastic model of resource consumption. Since resources in the rover domain are continuous, we use the continuous variables of a hybrid MDP to represent resources. (Note that 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.) Resource constraints are represented in the form of executability constraints on actions, where $A_n(\mathbf{x})$ denotes the set of actions executable in state (n, \mathbf{x}) . An action cannot be executed in a state that does not satisfy its minimum resource requirements.

Having discussed how to incorporate resource consumption and resource constraints in a hybrid MDP, we next discuss how to formalize *over-subscription planning*. In our rover planning problem, scientists provide the planner with a set of “goals” they would like the rover to achieve, where each goal corresponds to a scientific task such as taking a picture of a rock or performing an analysis of a soil sample. The scientists also specify a utility for each goal. Usually only a subset of these goals is feasible under resource constraints, and the problem is to find a feasible plan that maximizes expected utility. Over-subscription planning for planetary exploration rovers has been considered by Smith (2004) and van den Briel *et al.* (2004) for deterministic domains. We consider over-subscription planning in stochastic domains, especially domains with stochastic resource consumption. This requires construction of conditional plans in which the selection of goals to achieve can change depending on resource availability.

In over-subscription planning, the utility associated with each goal can be achieved only once; no additional utility is achieved of repeating the task. Therefore, the discrete state must include a set of Boolean variables to keep track of the set of goals achieved so far by the rover, with one Boolean variable for each goal. Keeping track of already-achieved goals ensures a Markovian reward structure, since achievement of a goal is rewarded only if it was not achieved in the past. However, it also significantly increases the size of the discrete state space.¹

1. Maintaining history information to ensure a Markovian reward structure is related to work on planning with non-Markovian rewards, and provides a simple example of this more general class of problems. (Thiebaux, Gretton, Slaney, Price, & Kabanza, 2006).

2.3 Optimality equation

The rover planning problem we consider is a special case of a finite-horizon hybrid MDP. The Bellman optimality equation for this problem takes the following form:

$$\begin{aligned}
 V_n(\mathbf{x}) &= 0 \text{ when } (n, \mathbf{x}) \text{ is a terminal state,} \\
 V_n(\mathbf{x}) &= \max_{a \in A_n(\mathbf{x})} \left[\sum_{n' \in \mathcal{N}} \Pr(n' | n, \mathbf{x}, a) \int_{\mathbf{x}'} \Pr(\mathbf{x}' | n, \mathbf{x}, a, n') (R_{n'}(\mathbf{x}') + V_{n'}(\mathbf{x}')) d\mathbf{x}' \right]. \quad (1)
 \end{aligned}$$

We define a *terminal state* as a state in which no actions are eligible to execute, that is, $A_n(\mathbf{x}) = \emptyset$. We use terminal states to model various conditions for plan termination. This includes the situation in which all goals have been achieved; the situation in which resources have been exhausted; and the situation in which an action has resulted in some error condition that requires executing a safe sequence by the rover and terminating plan execution. In addition to terminal states, we assume an explicit *initial state* denoted (n_0, \mathbf{x}_0) .²

Assuming that resources are limited and non-replenishable, and that every action consumes some resource (and the amount consumed is greater than or equal to some positive quantity c), plan execution must terminate after a finite number of steps. The maximum number of steps is bounded by the initial resource allocation divided by c , the minimal resource consumption per step. The actual number of steps is usually much less and indefinite, because resource consumption is stochastic and because the choice of action influences resource consumption. Because the number of steps it takes for a plan to terminate is bounded but indefinite, we call this a *bounded-horizon MDP* in contrast to a finite-horizon MDP. However, we note that any bounded-horizon MDP can be converted to a finite-horizon MDP by specifying a horizon that is equal to the maximum number of plan steps, and introducing a no-op action that is taken in any terminal state.

Note that there is usually a difference between the number of plan steps and the time a plan takes to execute. 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. As a result, the time it takes to execute a plan step is both state and action dependent, and stochastic.

Given a hybrid MDP with a set of terminal states and an initial state (n_0, \mathbf{x}_0) , the objective is to find a policy $\pi : (\mathcal{N} \times \mathbf{X}) \rightarrow \mathcal{A}$ that maximizes expected cumulative reward. In our framework, cumulative reward is equal to the sum of rewards for the goals achieved before reaching a terminal state. 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 do not need to formulate this as a problem of multi-objective optimization, and we stay in a standard decision-theoretic framework. An optimal policy has a value function that satisfies the optimality equation given by Equation (1). In the rest of the paper, we discuss how to find such a policy.

2.4 Dynamic programming for structured hybrid MDPs

Because the planning problem we consider is a finite-horizon hybrid MDP, it can be solved by any algorithm for solving finite-horizon hybrid MDPs. Most algorithms for solving hybrid (and continuous-state) MDPs rely on some form of approximation. A widely-used approach is to discretize the continuous state space into a finite number of grid points and solve the resulting finite-state MDP using dynamic programming and interpolation (Rust, 1997; Munos & Moore, 2002). Another approach is parametric function approximation; a function associated with the dynamic programming problem – such as the value function or policy function – is approximated by a smooth function of k unknown parameters. In general, parametric function approximation is faster than grid-based approximation, but has the drawback that it may fail to converge, or else converge to an incorrect solution. Parametric function approximation is used by other algorithms for solving continuous-state MDPs besides dynamic programming. Reinforcement learning algorithms use artificial neural networks as function approximators (Bertsekas & Tsitsiklis, 1996), and an approach to solving MDPs called approximate linear programming has recently been extended to allow continuous state variables (Kveton, Hauskrecht, & Guestrin, 2006).

2. Our framework also allows an uncertain starting state, as long as its probability distribution is known.

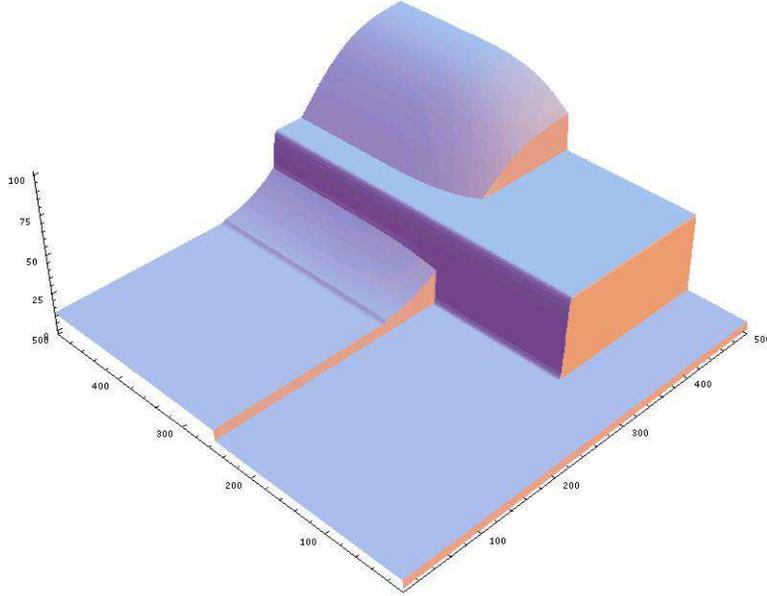


Figure 1: Value function in the initial state of a simple rover problem: optimal expected return as a function of two continuous variables (time and energy remaining).

We review another approach to solving hybrid (or continuous-state) MDPs that assumes the problem has special structure that can be exploited by the dynamic programming algorithm. The structure assumed by this approach ensures that the convolution $\int_{\mathbf{x}'} \Pr(\mathbf{x}' | n, \mathbf{x}, a, n') (R_{n'}(\mathbf{x}') + V_{n'}(\mathbf{x}')) d\mathbf{x}'$ in Equation (1) can be computed exactly in finite time, and the value function computed by dynamic programming is piecewise-constant or piecewise-linear. The initial idea for this approach can be traced to the work of Boyan and Littman (2000), who describe a class of MDPs called *time-dependent MDPs*, in which transitions take place along a single, irreversible continuous dimension. They describe a dynamic programming algorithm for computing an exact piecewise-linear value function when the transition probabilities are discrete and rewards are piecewise linear. Feng et al. (2004) extend this approach to continuous state spaces of more than one dimension, and consider MDPs with discrete transition probabilities and two types of reward models: piecewise constant and piecewise linear. Li and Littman (2005) further extend the approach to allow transition probabilities that are piecewise-constant, instead of discrete, although this extension requires some approximation in the dynamic programming algorithm.

The problem structure exploited by these algorithms is characteristic of the Mars rover domain and other over-subscription planning problems. Figure 1 shows the optimal value from the initial state of a typical Mars rover problem as a function of two continuous variables: the time and energy remaining (Bresina et al., 2002). The value function features a set of humps and plateaus, each of them representing a region of the state space where similar goals are pursued by the optimal policy. The sharpness of a hump or plateau reflects uncertainty about achieving the goal(s). Constraints that impose minimal resource levels before attempting some actions introduce sharp cuts in the regions. The vast plateau regions where the expected reward is nearly constant represent regions of the state space where the optimal policy is the same, and the probability distribution over future histories induced by this optimal policy is nearly constant.

The structure in such a value function is exploited by partitioning the continuous state space into a finite number of hyper-rectangular regions. (A region is a (hyper-)rectangle if it is the Cartesian product of intervals at each dimension.) In each hyper-rectangle, the value function is either constant (for a piecewise-constant function) or linear (for a piecewise-linear function). The resolution of the hyper-rectangular partitioning is adjusted to fit the value function. Large hyper-rectangles are used to represent large plateaus. Small hyper-rectangles are used to represent regions of the state space where a finer discretization of the value function is useful, such as the edges of plateaus and the curved hump where there is more time and energy available. A natural choice of data structures for rectangular partitioning of a continuous space is kd-trees (Friedman, Bentley, & Finkel, 1977), although other choices are possible.

The continuous-state domains of the transition and reward functions are similarly partitioned into hyper-rectangles. The reward function of each action has the same piecewise-constant (or piecewise-linear) representation as the value function. The transition function partitions the state space into regions for which the set of outcomes of an action and the probability distribution over the set of outcomes are identical. Following Boyan and Littman (2000), both relative and absolute transitions are supported. A relative outcome can be viewed as shifting a region by a constant δ . That is, for any two states x and y in the same region, the transition probabilities $Pr(x'|x, a)$ and $Pr(y'|y, a)$ are defined in terms of the probability of δ , such that $\delta = (x' - x) = (y' - y)$. An absolute outcome maps all states in a region to a single state. That is, for any two states x and y in the same region, $Pr(x'|x, a) = Pr(x'|y, a)$. We can view a relative outcome as a pair (δ, p) , where p is the probability of that outcome, and an absolute outcome as a pair (x', p) . This assumes there is only a finite number of non-zero probabilities, i.e., the probability distribution is discretized, which means that for any state and action, a finite set of states can be reached with non-zero probability. This representation guarantees that a dynamic programming update of a piecewise-constant value function results in another piecewise-constant value function. Feng et al. (2004) show that for any finite horizon, there exists a partition of the continuous space into hyper-rectangles over which the optimal value function is piecewise constant or linear.

The restriction to discrete transition functions is a strong one, and often means the transition function must be approximated. For example, rover power consumption is normally distributed. Any continuous transition function can be approximated by an appropriately fine discretization, and Feng et al. (2004) argue that this provides an attractive alternative to function approximation approaches in that it approximates the model but then solves the approximate model exactly, rather than finding an approximate value function for the original model. (For this reason, we will sometimes refer to finding optimal policies and value functions, even when the model has been approximated.) To avoid discretizing the transition function, Li and Littman (2005) describe an algorithm that allows piecewise-constant transition functions, in exchange for some approximation in the dynamic programming algorithm. We refer to the work of Feng et al. (2004) and Li and Littman (2005) for details of these algorithms.

3. Heuristic search in hybrid state spaces

In this section, we present our main contribution: an approach to solving finite-horizon hybrid MDPs that uses a novel generalization of the heuristic search algorithm AO*. In particular, we describe a generalization of this algorithm for solving hybrid MDPs in which the continuous variables represent consumable resources.

The motivation for using heuristic search is the potentially *huge* size of the state space, which makes dynamic programming 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. As others have shown, AO* can be very effective in solving planning problems that have a large state space because it only considers states that are reachable from an initial state, and uses an informative heuristic function to focus on states that are reachable in the course of executing a good plan. As a result, AO* can often find an optimal plan by exploring a small fraction of the entire state space.

We begin this section with a review of the standard AO* algorithm. Then we consider how to generalize AO* to search in hybrid state spaces and discuss the properties of the generalized algorithm, as well as how to implement it in an efficient way.

3.1 AO*

Recall that AO* is an algorithm for AND/OR graph search problems (Nilsson, 1980; Pearl, 1984). 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. Hansen and Zilberstein (2001) describe how AND/OR graph search techniques can be used in solving MDPs.

Following Nilsson (1980) and Hansen and Zilberstein (2001), we define an AND/OR graph as a hypergraph. Instead of arcs that connect pairs of nodes as in an ordinary graph, a hypergraph has *hyperarcs*, or *k-connectors*, that connect a node to a set of k successor nodes. When an MDP is represented by a hypergraph, each node corresponds to a state; the root node corresponds to the start state, and the leaf nodes correspond to terminal states. Thus we often use the word state to refer to the corresponding node in the hypergraph representing an MDP. A k -connector corresponds to an action that transforms a state into one of k possible successor states, with a probability attached to each successor such that the probabilities sum to one. In this section, we assume the AND/OR graph is acyclic, which is consistent with our assumption that the underlying MDP has a bounded-horizon.

In AND/OR graph search, a solution takes the form of an acyclic subgraph called a *solution graph*, which is defined as follows:

- the start node belongs to a solution graph;
- for every non-terminal node in a solution graph, exactly one outgoing k -connector (corresponding to an action) is part of the solution graph and each of its successor nodes also belongs to the solution graph;
- every directed path in the solution graph terminates at a terminal node.

A solution graph that maximizes expected cumulative reward is found by solving the following system of equations,

$$V^*(s) = \begin{cases} 0 & \text{if } s \text{ is a terminal state,} \\ \max_{a \in A(s)} [\sum_{s' \in S} Pr(s'|s, a) (R(s') + V^*(s'))] & \text{otherwise,} \end{cases} \quad (2)$$

where $V^*(s)$ denotes the expected value of an optimal solution for state s and V^* is called the optimal *evaluation function*. Note that this is identical to the optimality equation for hybrid MDPs defined in Equation (1), if the latter is restricted to a discrete state space. In keeping with the convention in the literature on MDPs, we treat this as a value-maximization problem even though AO* is usually formalized as solving a cost-minimization problem.

For state-space search problems that are formalized as AND/OR graphs, an optimal solution graph can be found using the heuristic search algorithm AO* (Nilsson, 1980; Pearl, 1984). Like other heuristic search algorithms, the advantage of AO* over dynamic programming is that it can find an optimal solution without evaluating all problem states. Therefore, a graph is not usually supplied explicitly to the search algorithm. An implicit graph, G , is specified implicitly by a start node or start state s and a successor function that generates the successors states for any state-action pair. The search algorithm constructs an *explicit graph*, G' , that initially consists only of the start state. A tip or leaf state of the explicit graph is said to be terminal if it is a goal state (or some other state in which an action cannot be taken); otherwise, it is said to be nonterminal. A nonterminal tip state can be *expanded* by adding to the explicit graph its outgoing k -connectors (one for each action) and any successor states not already in the explicit graph.

AO* solves a state-space search problem by gradually building a solution graph, beginning from the start state. A *partial solution graph* is defined similarly to a solution graph, with the difference that tip states of a partial solution graph may be nonterminal states of the implicit AND/OR graph. A partial solution graph is defined as follows:

1. The explicit graph G' initially consists of the start state s .
2. While the best solution graph has some nonterminal tip state:
 - (a) *Expand best partial solution*: Expand some nonterminal tip state s of the best partial solution graph and add any new successor states to G' . For each new state s' added to G' by expanding s , if s' is a terminal state then $V(s') := 0$; else $V(s') := H(s')$.
 - (b) *Update state values and mark best actions*:
 - i. Create a set Z that contains the expanded state and all of its ancestors in the explicit graph along marked action arcs. (*I.e.*, only include ancestor states from which the expanded state can be reached by following the current best solution.)
 - ii. Repeat the following steps until Z is empty.
 - A. Remove from Z a state s such that no descendent of s in G' occurs in Z .
 - B. Set $V(s) := \max_{a \in A(s)} \sum_{s'} Pr(s'|s, a) (R(s') + V(s'))$ and mark the best action for s . (When determining the best action resolve ties arbitrarily, but give preference to the currently marked action.)
 - (c) *Identify the best solution graph and all nonterminal states on its fringe*
3. Return an optimal solution graph.

Table 1: AO* algorithm.

- the start state belongs to a partial solution graph;
- for every non-tip state in a partial solution graph, exactly one outgoing k -connector (corresponding to an action) is part of the partial solution graph and each of its successor states also belongs to the partial solution graph;
- every directed path in a partial solution graph terminates at a tip state of the explicit graph.

The value of a partial solution graph is defined similarly to the value of a solution graph. The difference is that if a tip state of a partial solution graph is nonterminal, it does not have a value that can be propagated backwards. Instead, we assume there is an admissible heuristic estimate $H(s)$ of the maximal-value solution graph for state s . A heuristic evaluation function H is said to be *admissible* if $H(s) \geq V^*(s)$ for every state s . We can recursively calculate an admissible heuristic estimate $V(s)$ of the optimal value of any state s in the explicit graph as follows:

$$V(s) = \begin{cases} 0 & \text{if } s \text{ is a terminal state} \\ H(s) & \text{if } s \text{ is a nonterminal tip state} \\ \max_{a \in A(s)} [\sum_{s' \in S} Pr(s'|s, a) (R(s') + V(s'))] & \text{otherwise.} \end{cases} \quad (3)$$

The best partial solution graph can be determined at any time by propagating heuristic estimates from the tip states of the explicit graph to the start state. If we mark the action that maximizes the value of each state, the best partial solution graph can be determined by starting at the root of the graph and selecting the best (*i.e.*, marked) action for each reachable state.

Table 1 outlines the algorithm AO* for finding an optimal solution graph in an acyclic AND/OR graph. It interleaves forward expansion of the best partial solution with a value update step that updates estimated state values and the best partial solution. In the simplest version of AO*, the values of the expanded state and all of its ancestor states in the explicit graph are updated. In fact, the only ancestor states that need to be re-evaluated are those from which the expanded state can be reached by taking marked actions (*i.e.*, by choosing the best action for each state). Thus, the parenthetical remark in step 2(b)i of Table 1 indicates

that a parent s' of state s is not added to Z unless both the estimated value of state s has changed, and state s can be reached from state s' by choosing the best action for state s' . HAO* terminates when the policy expansion step does not find any nonterminal states on the fringe of the best solution graph. At this point, the best solution graph is an optimal solution.

Following the literature on AND/OR graph search, we have so far referred to the solution found by AO* as a solution graph. But in the following, when AO* is used to solve an MDP, we sometimes follow the literature on MDPs in referring to a solution as a policy, or as a policy graph, to indicate that a policy is represented in the form of a graph.

3.2 Hybrid AO*

We now consider how to generalize AO* to solve a bounded-horizon hybrid MDP. The challenge we face in applying AO* to this problem is the challenge of performing state-space search in a continuous state space.

The solution we adopt is to search in an *aggregate state space* that is represented by a AND/OR graph in which there is a node for each distinct value of the discrete component of the state. In other words, each node of the AND/OR graph represents a region of the continuous state space in which the discrete value is the same. Given this partition of the continuous state space, we use AND/OR graph search techniques to solve the MDP for those parts of the state space that are reachable from the start state under the best policy.

However, AND/OR graph search techniques must be modified in important ways to allow search in a hybrid state space that is represented in this way. In particular, there is no longer a correspondence between the nodes of the AND/OR graph and individual states. Each node now corresponds to a continuous region of the state space, and different actions may be optimal for different hybrid states associated with the same search node. In the case of rover planning, for example, the best action is likely to depend on how much energy or time is remaining, and energy and time are continuous state variables.

To address this problem and still find an optimal solution, we attach to each search node a set of functions (of the continuous variables) that make it possible to associate different values, heuristics, and actions with different hybrid states corresponding to the same search node. As before, the *explicit search graph* consists of all nodes and edges of the AND/OR graph that have been generated so far, and describes all the states that have been considered so far by the search algorithm. The difference is that we use a more complex state representation in which a set of continuous functions allows representation and reasoning about the continuous part of the state space associated with a search node.

We begin by describing this more complex node data structure, and then describe the HAO* algorithm.

3.2.1 DATA STRUCTURES

Each node n of the explicit AND/OR graph G' consists of the following:

- The value of the discrete state variable.
- Pointers to its parents and children in the explicit graph and the policy graph.
- $Open_n(\cdot) \rightarrow \{0, 1\}$: – the “Open list”. For each $\mathbf{x} \in \mathbf{X}$, $Open_n(\mathbf{x})$ indicates whether (n, \mathbf{x}) is on the frontier of the explicit graph, i.e., generated but not yet expanded.
- $Closed_n(\cdot) \rightarrow \{0, 1\}$: – the “Closed list”. For each $\mathbf{x} \in \mathbf{X}$, $Closed_n(\mathbf{x})$ (a state cannot be both open and closed) indicates whether (n, \mathbf{x}) is in the interior of the explicit graph, i.e., already expanded.

Note that, for all (n, \mathbf{x}) , $Open_n(\mathbf{x})Closed_n(\mathbf{x}) = 0$ (a state cannot be both open and closed). There can be parts of the continuous state space associated with a node that are neither open nor closed. Until the explicit graph contains a trajectory from the start state to a particular hybrid state, that hybrid state is not considered generated, even if the search node to which it corresponds has been generated; such states are neither open nor closed. In addition, only non-terminal states can be open or closed. Note that we don't refer to open or closed nodes; instead, we refer to the hybrid states associated with nodes as being open or closed.

- $H_n(\cdot)$ – the heuristic function. For each $\mathbf{x} \in \mathbf{X}$, $H_n(\mathbf{x})$ is a heuristic estimate of the optimal expected cumulative reward from state (n, \mathbf{x}) .
- $V_n(\cdot)$ – the value function. For any open state (n, \mathbf{x}) , $V_n(\mathbf{x}) = H_n(\mathbf{x})$. For any closed state (n, \mathbf{x}) , $V_n(\mathbf{x})$ is obtained by backing up the values of its successor states, as in Equation (4).
- $\pi_n(\cdot) \rightarrow A$: – the policy. Note that it is defined for closed states only.
- $Reachable_n(\cdot) \rightarrow \{0, 1\}$: For each $\mathbf{x} \in \mathbf{X}$, $Reachable_n(\mathbf{x})$ indicates whether (n, \mathbf{x}) is reachable by executing the current best policy beginning from the start state (n_0, \mathbf{x}_0) .

We assume that these various continuous functions, which represent information about the hybrid states associated with a search node, partition the state space associated with a node into a discrete number of *regions*, and associate a distinct value or action with each region. Given such a partitioning, the HAO* algorithm expands and evaluates these regions of the hybrid state space, instead of individual hybrid states. The finiteness of the partition is important to ensure that the search frontier can be extended by a finite number of expansions, and to ensure that HAO* can terminate after a finite number of steps. In our implementation of HAO*, described in Section 4, we use the piecewise-constant partitioning of a continuous state space used by Feng et al. (2004). However, any method of discrete partitioning could be used, provided that the condition above holds (see for instance (Li & Littman, 2005)). Note that two forms of state-space partitioning are used in our algorithm. First, the hybrid state space is partitioned into a finite number of regions, one for each discrete state, where each of these regions corresponds to a node of the AND/OR graph. Second, the continuous state space associated with a particular node is further partitioned into smaller regions based on a piecewise-constant representation of a continuous function, such as the one used by Feng et al. (2004).

In addition to this more complex representation of the nodes of an AND/OR graph, our algorithm requires a more complex definition of the the best (partial) solution. In standard AO*, the one-to-one correspondence between nodes and individual states means that a solution or policy can be represented entirely by a graph, called the (partial) solution graph, in which a single action is associated with each node. In the HAO* algorithm, a continuum of states is associated with each node, and different actions may be optimal for different regions of the state space associated with a particular node. For the HAO* algorithm, a (partial) solution graph is a sub-graph of the explicit graph that is defined as follows:

- the start node belongs to a solution graph;
- for every non-tip node in a solution graph, one *or more* outgoing k -connectors are part of the solution graph, one for each action that is optimal for some hybrid state associated with the node, and each of their successor nodes also belongs to the solution graph;
- every directed path in the solution graph terminates at a tip node of the explicit graph.

The key difference in this definition is that there may be more than one optimal action associated with a node, since different actions may be optimal for different hybrid states associated with the node. A policy is represented not only by a solution graph, but by the continuous functions $\pi_n(\cdot)$ and $Reachable_n(\cdot)$. In particular, a (partial) policy π specifies an action for each reachable region of the continuous state space. The best (partial) policy is the one that satisfies the following optimality equation:

$$\begin{aligned}
 V_n(\mathbf{x}) &= 0 \text{ when } (n, \mathbf{x}) \text{ is a terminal state,} \\
 V_n(\mathbf{x}) &= H_n(\mathbf{x}) \text{ when } (n, \mathbf{x}) \text{ is a nonterminal open state,} \\
 V_n(\mathbf{x}) &= \max_{a \in A_n(\mathbf{x})} \left[\sum_{n' \in N} \Pr(n' | n, \mathbf{x}, a) \int_{\mathbf{x}'} \Pr(\mathbf{x}' | n, \mathbf{x}, a, n') (R_{n'}(\mathbf{x}') + V_{n'}(\mathbf{x}')) d\mathbf{x}' \right]. \quad (4)
 \end{aligned}$$

Note that this optimality equation is only satisfied for regions of the state space that are reachable from the start state, (n_0, \mathbf{x}_0) by following an optimal policy.

1. The explicit graph G' initially consists of the start node and corresponding start state (n, \mathbf{x}) , marked as open and reachable.
2. While $Reachable_n(\mathbf{x}) \cap Open_n(\mathbf{x})$ is non-empty for some (n, \mathbf{x}) :
 - (a) *Expand best partial solution*: Expand one or more region(s) of open states on the frontier of the explicit state space that is reachable by following the best partial policy. Add new successor states to G' . In some cases, this requires adding a new node to the AND/OR graph. In other cases, it simply involves marking one or more regions of the continuous state space associated with an existing node as open. States in the expanded region(s) are marked as closed.
 - (b) *Update state values and mark best actions*:
 - i. Create a set Z that contains the node(s) associated with the just expanded regions of states and all ancestor nodes in the explicit graph along marked action arcs.
 - ii. Decompose the part of the explicit AND/OR graph that consists of nodes in Z into strongly connected components.
 - iii. Repeat the following steps until Z is empty.
 - A. Remove from Z a set of nodes such that (1) they all belong to the same connected component, and (2) no descendent of these nodes occurs in Z .
 - B. For every node n in this connected component and for all states (n, \mathbf{x}) in any expanded region of node n , set
$$V_n(\mathbf{x}) := \max_{a \in A_n(\mathbf{x})} \left[\sum_{n' \in N} \Pr(n' | n, \mathbf{x}, a) \int_{\mathbf{x}'} \Pr(\mathbf{x}' | n, \mathbf{x}, a, n') (R_{n'}(\mathbf{x}') + V_{n'}(\mathbf{x}')) d\mathbf{x}' \right],$$
and mark the best action. (When determining the best action resolve ties arbitrarily, but give preference to the currently marked action.) Repeat until there is no longer a change of value for any of these nodes.
 - (c) *Identify the best solution graph and all nonterminal states on its frontier*. This step updates $Reachable_n(\mathbf{x})$.
3. Return an optimal policy.

Table 2: HAO* algorithm.

3.2.2 ALGORITHM

Table 2 gives a high-level summary of the HAO* algorithm. In outline, it is the same as the AO* algorithm, and consists of iteration of the same three steps; solution (or policy) expansion, use of dynamic programming to update the current value function and policy, and analysis of reachability to identify the frontier of the solution that is eligible for expansion. In detail, it is modified in several important ways to allow search of a hybrid state space. In the following, we discuss the modifications to each of these three steps.

Policy expansion All nodes of the current solution graph are identified and one or more open regions associated with these nodes are selected for expansion. That is, one or more regions of the hybrid state space in the intersection of Open and Reachable is chosen for expansion. All actions applicable to the states in these open regions are simulated, and the results of these actions are added to the explicit graph. In some cases, this means adding a new node to the AND/OR graph. In other cases, it simply involves marking one or more regions of the continuous state space associated with an existing node as open. More specifically, when an action leads to a new node, this node is added to the explicit graph, and all states corresponding to this

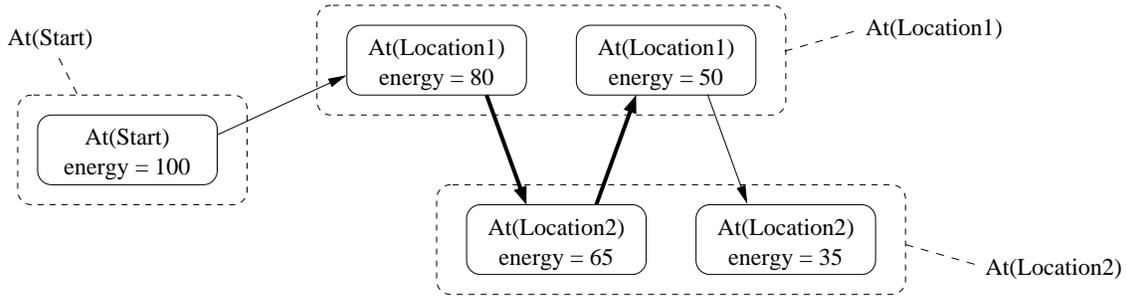


Figure 2: Phantom loops in HAO*: solid boxes represent Markov states. Dashed boxes represent search nodes, that is, the projection of Markov states on the discrete components. Arrows represent possible state transition. Bold arrows show an instance of phantom loop in the search space.

node that are reachable from the expanded region(s) after the action under consideration are marked as open. When an action leads to an existing node, any region(s) of Markov states in this node that is both reachable from the expanded region(s) and not marked as closed is marked open. Expanded regions of the state space are marked as closed. Thus, different regions associated with the same node can be opened and expanded at different times.

Dynamic programming As in standard AO*, the value of any newly expanded node n must be updated by computing a Bellman backup based on the value functions of the children of n in the explicit graph. For each expanded region of the state space associated with node n , each action is evaluated, the best action is selected, and the corresponding continuous value function is associated with the region. The continuous-state value function is computed by evaluating the continuous integral in Equation (4). We can use any method for computing this integral. In our implementation, we use the dynamic programming algorithm of Feng et al. (2004). As reviewed in Section 2.4, they show that the continuous integral over \mathbf{x}' can be computed exactly, as long as the transition and reward functions satisfy certain conditions. Note that as values are updated, the division of the continuous space associated with a node may be revised.

Once the expanded regions of the continuous state space associated with a node n are re-evaluated, the new values must be propagated backward (i.e. upward) in the explicit graph. The backward propagation stops at nodes where the value function is not modified, or at the root node. The standard AO* algorithm, summarized in Figure 1, assumes that the AND/OR graph in which it searches is acyclic. There are extensions of AO* for searching in AND/OR graphs that contain cycles. One line of research is concerned with how to find acyclic solutions in AND/OR graphs that contain cycles (Jimenez & Torras, 2000). Another generalization of AO*, called LAO*, allows solutions to contain “loops” in order to specify policies for infinite-horizon MDPs (Hansen & Zilberstein, 2001).

Given our assumption that every action has positive resource consumption, there can be no loops in the state space of our problem because the resources available decrease at each step. However, there can be loops in the AND/OR graph. This is possible because the AND/OR graph represents a projection of the state space onto a smaller space that consists of only the discrete component of the state. For example, it is possible for the rover to return to the same site it has visited before. The rover is not actually in the same state, since it has fewer resources available. But the AND/OR graph represents a projection of the state space that does not include the continuous aspects of the state, such as resources, and this means the rover can visit a state that projects to the same node of the AND/OR graph as a state it visited earlier, as shown in Figure 2. As a result, there can be loops in the AND/OR graph, and even loops in the part of the AND/OR graph that corresponds to a solution. But in a sense, these are “phantom loops” that can only appear in the projected state space, and not in the real state space.

Nevertheless we must modify the dynamic programming (DP) algorithm to deal with these loops. Because there are no loops in the real state space, we know that the exact value function can be updated by a finite number of backups performed in the correct order, with one backup performed for any state that can be visited along a path from the start state to the expanded node(s). But because multiple states can map to the same AND/OR graph node, the continuous region of the state space associated with a particular node may need to be evaluated more than once. To identify the AND/OR graph nodes that need to be evaluated more than once, we use the following two-step algorithm.

First, we consider the part of the AND/OR graph that consists of ancestor nodes of the just expanded node(s). This is the set Z of nodes identified at the beginning of the DP step. We decompose this part of the graph into strongly connected components. The graph of strongly connected components is acyclic and can be used to prescribe the order of backups in almost the same way as in the standard AO* algorithm. In particular, the nodes in a particular component are not backed up until all nodes in its descendent components have been backed up. Note that in the case of an acyclic graph, every strongly connected component has a single node. It is only possible for a connected component to have more than one node if there are loops in the AND/OR graph.

If there are loops in the AND/OR graph, the primary change in the DP step of the algorithm occurs when it is time to perform backups on the nodes in a connected component with more than one node. In this case, all nodes in the connected component are evaluated. Then they are repeatedly re-evaluated until the value functions of these nodes converge, that is, until there is no change in the values of any of the nodes. Because there are no loops in the real state space, convergence is guaranteed to occur after a finite number of steps. Typically, it occurs after a very small number of steps. An advantage of decomposing the AND/OR graph into connected components is that it identifies loops and localizes their effect to a small number of nodes. Typically, most nodes of the graph need to be evaluated just once during the DP step, and only a small number of nodes (and often none) need to be evaluated more than once.

Analysis of reachability Change in the value function can lead to change in the optimal policy, and thus a change in which states are visited by the best policy. This in turn can affect which open regions of the state space are eligible to be expanded. In this final step, HAO* identifies the best (partial) policy and recomputes $Reachable_n$ for all nodes and states in the explicit graph, as follows. For each node n in the best (partial) solution graph, consider each of its parents n' in the solution graph, and all the actions a that can lead from one of the parents to n . Then $Reachable_n(\mathbf{x})$ is the support of $P_n(\mathbf{x})$, where

$$P_n(\mathbf{x}) = \sum_{(n',a) \in \Omega_n} \int_{\mathbf{X}} Reachable_{n'}(\mathbf{x}') \Pr(n | n', \mathbf{x}', a) \Pr(\mathbf{x} | n', \mathbf{x}', a, n) d\mathbf{x}' , \quad (5)$$

that is, $Reachable_n(\mathbf{x}) = \{\mathbf{x} \in \mathbf{X} : P_n(\mathbf{x}) > 0\}$. In Equation (5), Ω_n is the set of pairs (n', a) where a is the best action in n' for some reachable resource level:

$$\Omega_n = \{(n', a) \in N \times A : \exists \mathbf{x} \in \mathbf{X}, P_{n'}(\mathbf{x}) > 0, \pi_{n'}(\mathbf{x}) = a, \Pr(n | n', \mathbf{x}, a) > 0\} .$$

It is clear that we can restrict our attention to state-action pairs in Ω_n , only.

By performing this reachability analysis, HAO* identifies the frontier of the state space that is eligible for expansion. HAO* terminates when this frontier is empty, that is, when it does not find any hybrid states in the intersection of *Reachable* and *Open*.

3.3 Convergence and error bounds

We next consider some of the theoretical properties of HAO*. First, under reasonable assumptions, we prove that HAO* converges to an optimal policy after a finite number of steps. Then we discuss how to use HAO* to find sub-optimal policies with error bounds.

The proof of convergence after a finite number of steps depends, among other things, on the assumption that a hybrid MDP has a finite branching factor. In our implementation, this means that for any region of the

state space that can be represented by a hyper-rectangle, the set of successor regions after an action can be represented by a finite set of hyper-rectangles. From this assumption and the assumption that the number of actions is finite, it follows that for every assignment n to the discrete variables, the set

$$\{\mathbf{x} | (n, \mathbf{x}) \text{ is reachable from the initial state using some fixed sequence of actions}\}$$

is the union of a finite number of open or closed hyper-rectangles. This assumption can be viewed as a generalization of the assumption of a finite branching factor in a discrete AND/OR graph upon which the finite convergence proof of AO* depends.

Theorem 1 *If the heuristic functions H_n are admissible (optimistic), all actions have positive resource consumptions, both continuous backups and action application are computable exactly in finite time, and the branching factor is finite, then:*

1. *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*;*
2. *HAO* terminates after a finite number of steps;*
3. *After termination, $V_n(\mathbf{x})$ is equal to the optimal expected return in (n, \mathbf{x}) , for all (n, \mathbf{x}) reachable under an optimal policy, i.e., $\text{Reachable}_n(\mathbf{x}) > 0$.*

Proof: (1) The proof is by induction. Every state (n, \mathbf{x}) is assigned an initial heuristic estimate, and $V_n(\mathbf{x}) = H_n(\mathbf{x}) \geq V_n^*(\mathbf{x})$ by the admissibility of the heuristic evaluation function. We make the inductive hypothesis that at some point in the algorithm, $V_n(\mathbf{x}) \geq V_n^*(\mathbf{x})$ for every state (n, \mathbf{x}) . If a backup is performed for any state (n, \mathbf{x}) ,

$$\begin{aligned} V_n(\mathbf{x}) &= \max_{a \in A_n(\mathbf{x})} \left[\sum_{n' \in N} \text{Pr}(n' | n, \mathbf{x}, a) \int_{\mathbf{x}'} \text{Pr}(\mathbf{x}' | n, \mathbf{x}, a, n') (R_{n'}(\mathbf{x}') + V_{n'}(\mathbf{x}')) d\mathbf{x}' \right] \\ &\geq \max_{a \in A_n(\mathbf{x})} \left[\sum_{n' \in N} \text{Pr}(n' | n, \mathbf{x}, a) \int_{\mathbf{x}'} \text{Pr}(\mathbf{x}' | n, \mathbf{x}, a, n') (R_{n'}(\mathbf{x}') + V_{n'}^*(\mathbf{x}')) d\mathbf{x}' \right] \\ &= V_n^*(\mathbf{x}), \end{aligned}$$

where the last equality restates the Bellman optimality equation.

(2) Because each action has positive resource consumption, and resources are finite and non-replenishable, the complete implicit AND/OR graph must be finite. For the same reason, this graph can be turned into a finite graph without loops: Along any directed loop in this graph, the amount of maximal available resources must decrease by some ϵ which is a positive lower-bound on the amount of resources consumed by an action. Each node in this graph may be expanded a number of times that is bounded by the number of its ancestor. (Each time a new ancestor is discovered, it may lead to an update in the set of reachable regions for this node.) Moreover, finite branching factor implies that the number of regions considered within each node is bounded (because there are finite ways of reaching this node, each of which contributes a finite number of hyper-rectangles). Thus, overall, the number of regions considered is finite, and the processing required for each region expansion is finite (because action application and backups are computed in finite time). This leads to the desired conclusion.

(3) The search algorithm terminates when the policy for the start state (n_0, \mathbf{x}_0) is complete, that is, when it does not lead to any unexpanded states. For every state (n, \mathbf{x}) that is reachable by following this policy, it is contradictory to suppose $V_n(\mathbf{x}) > V_n^*(\mathbf{x})$ since that implies a complete policy that is better than optimal. By the Bellman optimality equation of Equation (1), we know that $V_n(\mathbf{x}) \geq V_n^*(\mathbf{x})$ for every state in this complete policy. Therefore, $V_n(\mathbf{x}) = V_n^*(\mathbf{x})$. \square

HAO* not only converges to an optimal solution, stopping the algorithm early allows a flexible tradeoff between solution quality and computation time. 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 this assumption, the error of the greedy policy at each step of the algorithm can be bounded. We show this by using a decomposition of the value function described by Chakrabarti et al.(1988) and Hansen and Zilberstein (2001). We note that at any point in the algorithm, the value function can be decomposed into two parts, $g_n(\mathbf{x})$ and $h_n(\mathbf{x})$, such that

$$\begin{aligned} g_n(\mathbf{x}) &= 0 \text{ when } (n, \mathbf{x}) \text{ is an open state, on the fringe of the greedy policy; otherwise,} \\ g_n(\mathbf{x}) &= \sum_{n' \in N} \Pr(n' | n, \mathbf{x}, a^*) \int_{\mathbf{x}'} \Pr(\mathbf{x}' | n, \mathbf{x}, a^*, n') (R_n(\mathbf{x}) + g_{n'}(\mathbf{x}')) d\mathbf{x}', \end{aligned} \quad (6)$$

and

$$\begin{aligned} h_n(\mathbf{x}) &= H_n(\mathbf{x}) \text{ when } (n, \mathbf{x}) \text{ is an open state, on the fringe of the greedy policy; otherwise,} \\ h_n(\mathbf{x}) &= \sum_{n' \in N} \Pr(n' | n, \mathbf{x}, a^*) \int_{\mathbf{x}'} \Pr(\mathbf{x}' | n, \mathbf{x}, a^*, n') h_{n'}(\mathbf{x}') d\mathbf{x}', \end{aligned} \quad (7)$$

where a^* is the action that maximizes the right-hand side of Equation (4). Note that $V_n(\mathbf{x}) = g_n(\mathbf{x}) + h_n(\mathbf{x})$. We use this decomposition of the value function to bound the error of the best policy found so far, as follows.

Theorem 2 *At each step of the HAO* algorithm, the error of the current best policy is bounded by $h_{n_0}(\mathbf{x}_0)$.*

Proof: For any state (n, \mathbf{x}) in the explicit search space, a lower bound on its optimal value is given by $g_n(\mathbf{x})$, which is the value that can be achieved by the current policy when the *done* action is executed at all fringe states, and an upper bound is given by $V_n(\mathbf{x}) = g_n(\mathbf{x}) + h_n(\mathbf{x})$, as established in Theorem 1. It follows that $h_{n_0}(\mathbf{x}_0)$ bounds the difference between the optimal value and the current admissible value of any state (n, \mathbf{x}) , including the initial state (n_0, \mathbf{x}_0) . \square

Note that the error bound for the initial state is $h_{n_0}(\mathbf{x}_0) = H_{n_0}(\mathbf{x}_0)$ at the start of the algorithm; it decreases with the progress of the algorithm; and $h_{n_0}(\mathbf{x}_0) = 0$ when HAO* converges to an optimal solution.

3.4 Heuristic Function

The heuristic function H_n focuses the search on reachable states that are most likely to be useful. The more informative the heuristic, the more scalable the search algorithm. In our implementation of HAO* for the rover planning problem, which is described in detail in the next section, we consider two admissible heuristic functions. The first is very simple: it assigns to each node the sum of all rewards associated with goals that have not been achieved so far. Note that this heuristic function only depends on the discrete component of the state, and not on the continuous variables; that is, the function $H_n(\mathbf{x})$ is constant over all values of \mathbf{x} . It is obvious that this heuristic is admissible, since it represents the maximum additional reward that could be achieved by continuing plan execution. It is not as obvious that a heuristic this simple could be useful. However the experimental results we present in Section 4 show that it is.

We can create another heuristic function for HAO* by solving a relaxation of the planning problem. In this relaxation, we assume that transitions for the continuous variables are deterministic; that is, we assume $\Pr(\mathbf{x}' | n, \mathbf{x}, a, n') \in \{0, 1\}$. To create an admissible heuristic, we assume that actions consume the minimum amount of all possible consumptions of a resource in the stochastic model. To create a non-admissible heuristic that is usually more informative, we assume that actions always consume their mean resource consumption. Because it is usually infeasible to solve this relaxed planning problem for the entire state space, we solve it using HAO* guided by a simple heuristic, such as the one described above. The relaxed problem is much easier to solve using HAO* than the original problem because deterministic resource consumption makes updates of the value functions V_n much simpler. After the relaxed problem is solved, the value function V_n of a node n in the relaxed graph represents the heuristic function H_n of the associated node in the original problem graph. However, we have 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 the heuristic may not be computed for all states for which it is needed. 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, and this allows us to obtain a heuristic estimate for many more states. If we reach states for which no heuristic estimate was obtained using these initial values, we simply solve the relaxed problem starting with this initial state.

3.5 Node Expansion Policy

HAO* works correctly and converges to an optimal solution no matter which continuous region(s) of which node(s) is expanded at each iteration (step 2.a). But the quality of the solution may improve more quickly by using some “heuristics” to choose which region(s) on the fringe to expand next.

One simple strategy is to select a node and expand all continuous regions of this node that are open and reachable. In a preliminary implementation, we expand (the region of) the node that is most likely to be reached using the greedy policy. Changes in the value of these states will have the greatest effect on the value of earlier nodes. Implementing this strategy requires performing the additional work involved in maintaining the probability associated with each state. If such probabilities are available, one could also focus on expanding the most promising node, that is, the node where the integral of $H_n(\mathbf{x})$ times the probability over all values of \mathbf{x} is the highest, as described in (Mausam, Benazera, Brafman, Meuleau, & Hansen, 2005).

Hansen and Zilberstein (2001) observed that, in the case of LAO*, the algorithm is more efficient if we expand several nodes in the fringe before performing dynamic programming in the explicit graph. This is because the cost of performing the update of a node largely dominates the cost of expanding a node. If we expand only one node of the fringe at each iteration, we might have to perform more DP back-ups than if we expand several nodes with common ancestors before proceeding to DP. In the limit, we might want to expand all nodes of the fringe at each algorithm iteration. Indeed, this variant of LAO* proved the most efficient in (Hansen & Zilberstein, 2001).

In the case of LAO*, updates are expensive because of the loops in the implicit graph. In HAO*, the update of a node induces a call to the hybrid dynamic programming module for each open region of the node. Therefore, the same technique is likely to produce the same benefit.

Pursuing this idea, we allowed our algorithm to expand all nodes in the fringe and all their descendants up to a fixed depth at each iteration. We defined a parameter called the *expansion horizon* and denoted k to represent, loosely speaking, the number of times the whole fringe is expanded at each iteration. When $k = 1$, HAO* expands all open and reachable regions of all nodes in the fringe before recomputing the optimal policy. When $k = 2$, it expands all regions in the fringe and all their children before updating the policy. At $k = 3$ it also consider the grand-children of regions in the fringe, and so on. When k tends to infinity, the algorithm essentially performs an exhaustive search: it first expands the graph of all reachable nodes, then performs one pass of (hybrid) dynamic programming in this graph to determine the optimal policy. By balancing node expansion and update, the expansion horizon allows tuning the algorithm behavior from an exhaustive search to a more traditional heuristic search. Our experiments showed that a value of k between 5 and 10 is optimal to solve our hardest benchmark problems (see section 4).

4. Experimental Evaluation

We tested our implementation of HAO* using a realistic, real-size NASA simulation of a planetary rover. We begin this section with a description of the simulation models and the details of our implementation, and then present and analyze the results of our experiments.



Figure 3: The K9 rover was developed at the Jet Propulsion Laboratory and NASA Ames Research Center as a prototype for the Mars Exploration Rovers (*aka*, Spirit and Opportunity). It is used to test advanced rover software, including automated planners of the rover's activities.

4.1 Domain Model and Problem Instances

Our simulations use a model of the K9 rover (see Figure 3) developed for the Intelligent Systems (IS) demo at NASA Ames Research Center in October 2004 (Pedersen et al., 2005). This is a complex real-size model of the K9 rover that uses command names understandable by the rover's execution language, so that the plans produced by our algorithm can be directly executed by the rover. We did not simplify the NASA model in any way for our experiments.

In this domain, the autonomous rover must navigate in a planar graph representing its surroundings and the authorized navigation paths, and schedule observations to be performed on different rocks situated at different locations. Only a subset of its observational goals can be achieved in a single run due to limited resources, namely, time and energy. Each action has uncertain positive resource consumptions and a probability of failing.

A significant amount of uncertainty in the domain comes from the tracking mechanism used by the rover. In order to perform a measurement on some rock, the rover must be tracking the rock. To navigate along a path, it must be tracking one of the rocks that enables following this path. However, the rover may randomly lose track of some rocks while navigating along a path, and it has no way to re-acquire a rock it has lost track of. (The probability of losing track of a rock depends on the rock and the path followed.) Finally, the number of rocks tracked strongly influences the duration and resource consumption of navigate actions; the higher the number of rocks tracked, the more costly it is to navigate along a path. This is because the rover has to stop

problem name	rover locations	paths	rocks	goals	fluents	actions
Rover1	7	10	3	3	30	43
Rover2	7	11	4	5	41	56
Rover3	9	16	5	6	49	73
Rover4	11	20	5	6	51	81

Table 3: Size of benchmark rover problems.

problem name	discrete states	reachable discrete states	Markov states	reachable Markov states	actions in longest branch
Rover1	$\approx 1.07 \cdot 10^9$	613	$\approx 1.43 \cdot 10^{12}$	155110	29
Rover2	$\approx 2.20 \cdot 10^{12}$	5255	$\approx 2.92 \cdot 10^{15}$	1274515	34
Rover3	$\approx 5.63 \cdot 10^{14}$	20393	$\approx 7.49 \cdot 10^{17}$	3992209	43
Rover4	$\approx 2.25 \cdot 10^{15}$	22866	$\approx 2.99 \cdot 10^{18}$	4580306	43

Table 4: Size of benchmark rover problems (continued).

regularly to re-acquire each rock tracked. It creates an incentive to limit the number of rocks tracked by the rover given the set of goals it has chosen and the path it intends to follow. The decision to start tracking a rock must be made before the rover begins to move. Once the rover starts moving, it may lose track of some rocks, and this may cause it to reconsider the set of goals it will pursue and the route to get to the corresponding rocks. Note that the rover can also use a *trackstop* action to stop tracking a rock that is no longer necessary.

In the following, we report the results of experiments performed using four problem instances of this rover domain. The characteristics of these problem instances are displayed in Tables 3 and 4. Table 3 shows the size of the problems in terms of rover locations, paths, rocks and goals. A problem may contain more goals than rocks if several goals (measurements) are associated with the same rock. Table 3 also shows the total number of fluents (boolean state variables) and actions in each problem. Table 4 shows the size of the search space and the optimal policy for each problem instance. The total number of discrete states is two raised to the power of the number of fluents. It is a huge state space, however, only a limited number of states can be reached from the start state along a single trajectory, depending on the initial resource levels. The third column in Table 4 shows the number of reachable discrete states if the initial time and energy levels are set to their maximum value. The next column shows the number of Markov states the problem would contain if the continuous state variables were discretized using the same discretization procedure used to discretize action outcomes in HAO* (see Section 2.4). It represents the number of states that a dynamic programming algorithm that does not exploit reachability, such as Value Iteration, would have to handle to solve these problems. Next, the number of reachable Markov states is computed assuming the initial resource levels are set to their maximum possible value. It represents the size of the discrete graph that standard AO* would have to explore in order to solve these problems, if the continuous variables were discretized in the same naive way. Finally, the last column shows the length of the longest branch in the optimal policy when the initial resource levels are set to their maximum value. The largest of the four instances (that is, Rover4) is exactly the problem of the October 2004 IS demo. This is considered a very large rover problem. (For example, it is much larger than the problems faced by the MER rovers that never visit more than one rock in a single planning cycle.)

Table 4 shows that our domain is huge, but simple reachability based on resource availability makes huge difference. This is mostly because our planning domain, which is very close to the K9 execution language, does not allow many fluents to be true simultaneously. The number of reachable discrete states, and thus size of the graph to explore, may seem small compared to other discrete combinatorial problems solved by AI techniques. However, a continuous approximation of the two-dimensional back-up is necessary at each step of DP to explore this graph, making the largest of our problems unsolvable in reasonable time by most current computers. The results below show that a large portion of reachable states can be avoided by using HAO*.

4.2 Algorithm Parameters

We used the very simple heuristic described in section 3.4, augmented with a small amount of domain knowledge. The value $H_n(\mathbf{x})$ of a state (n, \mathbf{x}) is essentially equal to the sum of the utilities of all goals not yet achieved in n . However, if the rover has already moved and a certain rock is not being tracked in state n , then all goals requiring this rock to be tracked are not counted into the sum. This reflects the fact that, once the rover has moved, it cannot start tracking a rock any more, and thus all goals that require this rock to be tracked are unreachable. The resulting heuristic is admissible (i.e., it never underestimates the value of a state), and it is straightforward to compute. Note that it does not depend on the current resource levels, so that the functions $H_n(\mathbf{x})$ are constant over all values of \mathbf{x} . We experimented with more complex heuristics such as those described in section 3.4, but our best results in terms of execution time were obtained with this simple heuristic.

We used the variant of the algorithm described in Section 3.5, where a parameter k representing the number of times the whole fringe is expanded at each iteration of HAO* allows the behavior of the algorithm to be tuned from an exhaustive search to a heuristic search.

4.3 Simulation Results

4.3.1 EFFICIENCY OF PRUNING

In a first set of simulations, we try to evaluate the efficiency of heuristic pruning in HAO*, that is, the portion of the discrete search space that is spared from exploration though the use of admissible heuristics. For this purpose, we compare the number of discrete states that are reachable for a given resource level with the number of states created and expanded by HAO*. We also consider the number of nodes in the optimal policy found by the algorithm.

Results for the four benchmark problems are presented in Fig. 4 and 5. These curves are obtained by fixing one resource to its maximum possible value and varying the other from 0 to its maximum. Therefore, they represent problems where mostly one resource is constraining. These result show, notably, that a single resource is enough to constrain the reachability of the state space significantly.

Not surprisingly, problems become larger as the initial resources increase, because more discrete states become reachable. Despite the simplicity of the heuristic used, HAO* is able to by-pass a significant part of the search space. Moreover, the bigger the problem, the more leverage the algorithm can take from the simple heuristic.

These results are quite encouraging, but the number of nodes created and expanded does not always reflect solution time. Therefore, we examine the time it takes for HAO* to produce solutions.

4.3.2 SOLUTION TIME

Figure 6 and 7 show HAO* execution time for the same set of experiments. Perhaps surprisingly, these curves do not show the same monotonicity as the previous and seem to contain a significant amount of noise. It is remarkable that the computation time does not always increase with the initial levels of resource, although the search space is bigger. It shows that complexity is not always well measured by the size of the search space. Some problems with less available resource are actually more constrained than larger problems, and thus take more time to solve.

Execution time is more closely related to the complexity of the optimal policy. It depends on the amount of branching and, in some cases, an increase of resource may eliminate the need for branching. Figure 8 and 9 show the number of nodes and branches in the optimal policy found by the algorithm, as well as the number of goals pursued by this policy. It shows that the size and the optimal policy and, secondarily, its number of branches, may explain most of the peaks in the execution time curves.

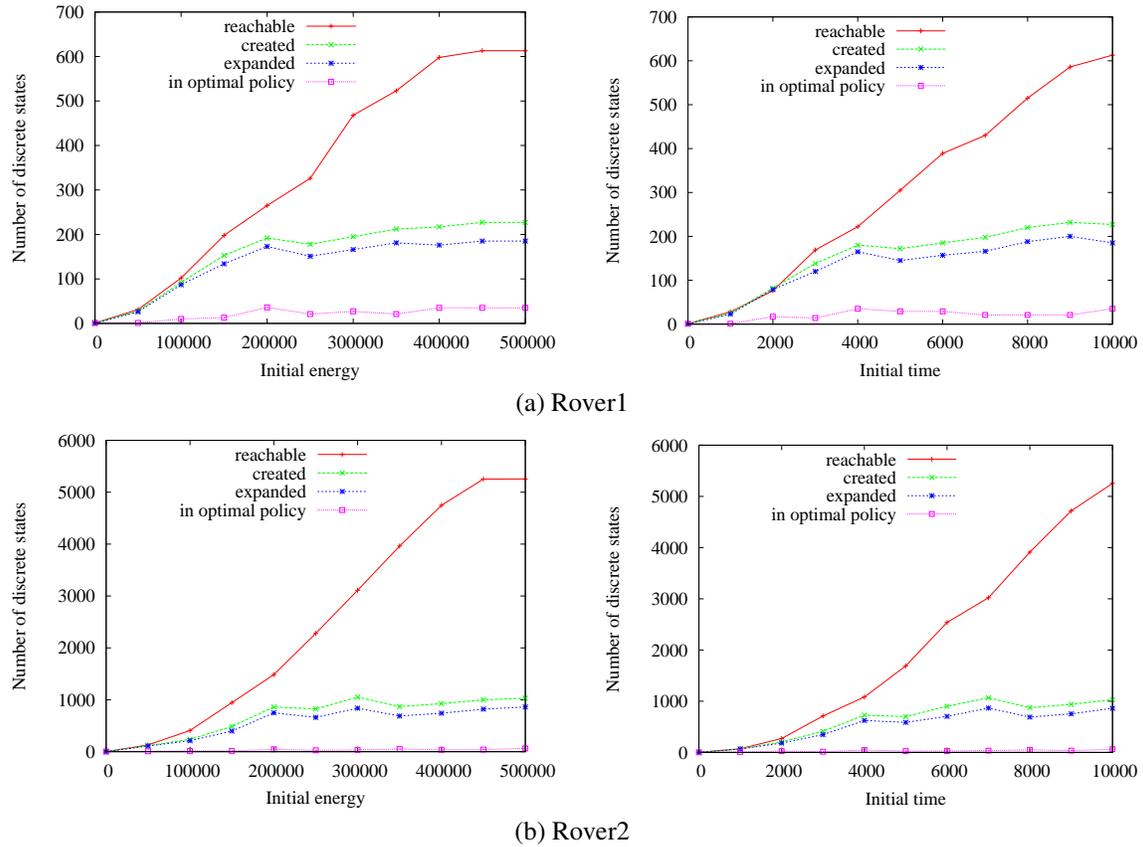


Figure 4: Number of nodes created and expanded by HAO* vs. number of reachable discrete states for two smallest benchmark problems. The graphs in the left column are obtained by fixing the initial time to its maximum value and varying the initial energy. The graphs in the right column are obtained by fixing the initial energy to its maximum value and varying the initial time. Results obtained with $k = 7$.

4.3.3 OPTIMAL EXPANSION HORIZON

Results of section 4.3.1 show that HAO* can leverage its simple heuristics to bypass a large portion of the search space. It does not mean however that HAO* will necessarily outperform a blind search algorithm that exhaustively expands the graph of reachable states, and then executes one-pass of DP in this graph to find the optimal policy. First, computation time shows large, seemingly arbitrary variations. Most importantly, as explained in Section 3.5, heuristic search expands a smaller graph than exhaustive search, but it must evaluate this graph much more often.

The expansion horizon was introduced in Section 3.5 to allow different trade-offs between the nodes expansion and node evaluation. We now study the influence of this parameter on the algorithm.

Figure 10 shows the evolution of the number of nodes created and expanded by HAO* as a function of the expansion horizon for the four benchmark problems. Not surprisingly, the algorithm visits an increasing number of nodes as the expansion horizon increases. It illustrates the fact that the algorithm tends to exhaustive search as k increases. For the two smallest problem instances, the experiment was stopped when k reached the length of the longest branch in the optimal policy (cf. Table 4). This guarantees that the op-

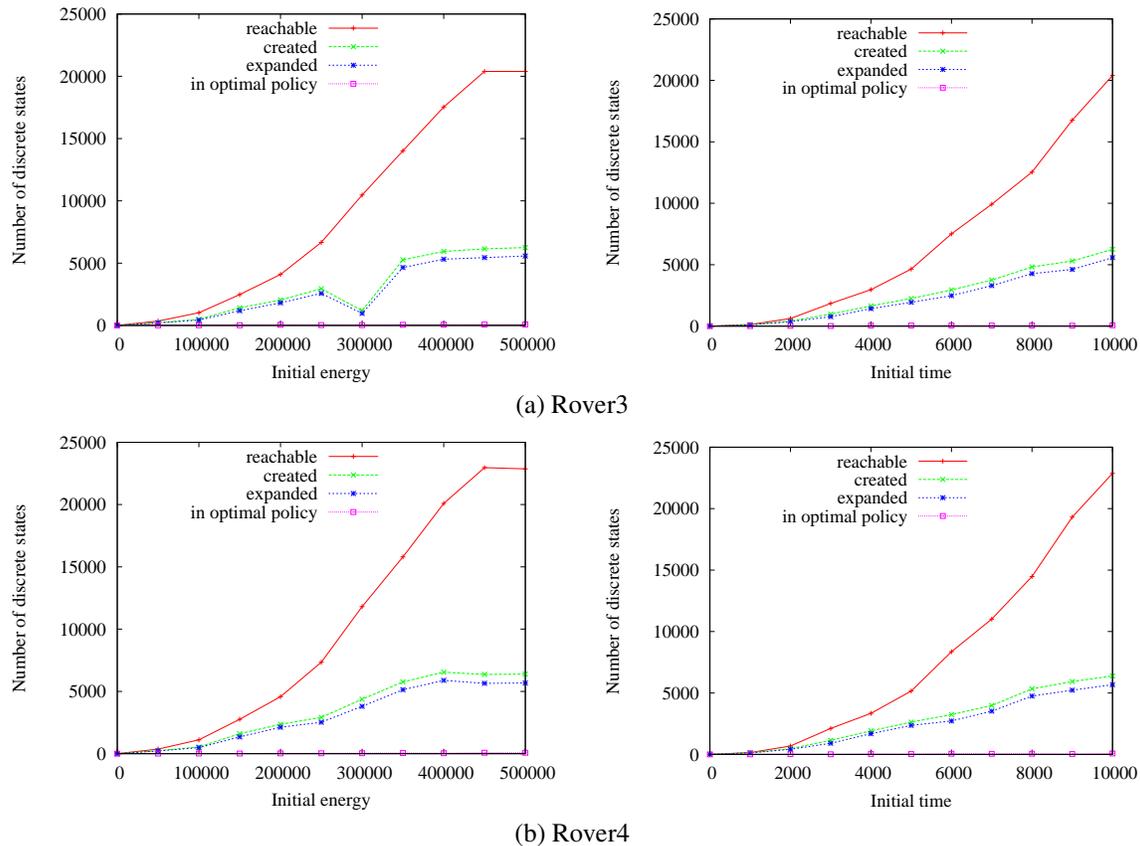


Figure 5: Same as Fig. 4 for the two largest benchmark problems.

timal solution is found at the first iteration. However, more iterations might be needed for the algorithm to complete (HAO* might still have to prove that this is an optimal solution). Nevertheless, the algorithm levels off before the end of the experience for the smallest problem. For the two largest problem instances, we had to interrupt the experiment when execution time became too large (see below). The similarities between the curves associated with these two instances is explained by the fact that Rover3 was derived from Rover4 (the IS demo problem) by deleting some rover locations and paths, and that the optimal policy at the fixed levels of resources does not traverse this part of the rover environment.

Next, we examine the evolution of HAO* execution time with the expansion horizon. Results are presented in Fig. 11. Again, execution time data appears more noisy than number of nodes. The first graph shows that, despite the lesser number of nodes explored, we cannot claim a clear advantage from using HAO* to solve the smallest problem (Rover1). As k increases, the algorithm levels-off and transforms into exhaustive search. The total time to perform this exhaustive exploration is not significantly larger than the performance of the best implementation of HAO*. The three largest problems clearly show the benefit of using HAO*. With Rover2, the algorithms levels-off around $k = 25$, but the duration of exhaustive search is significantly larger than the best implementation of HAO*. The benefits of HAO* appear most clearly with the two largest problem. Here, the algorithm is quickly overwhelmed by the combinatorial explosion of the search space as k increases. Simulations were interrupted when simulation time became too large.

Overall, our results show that there is a clear benefit in using admissible heuristics to prune the search space. However, a pure heuristic search is not the best solution, and a fine balance between node expansion

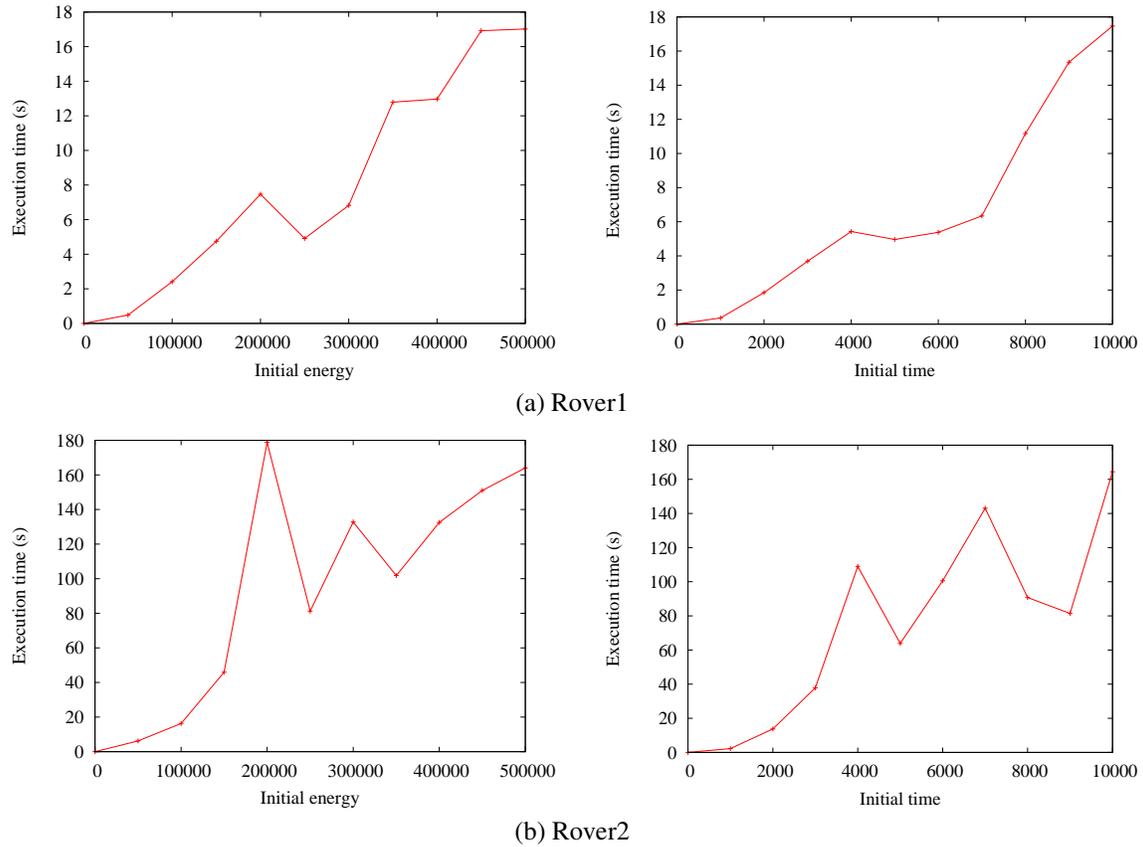


Figure 6: Execution time for the two smallest benchmark problems. The graphs in the left column are obtained by fixing the initial time to its maximum value, and the graphs in the right column are obtained by fixing the initial energy to its maximum. Results obtained with $k = 7$.

and node evaluation must be achieved. HAO* achieves efficient trade-offs to solve optimally large instances of the rover problem domain.

5. Conclusion

We have introduced a heuristic search approach to finding optimal conditional plans in domains characterized by continuous state variables that represent limited, consumable resources. Our HAO* algorithm is a variant of the AO* algorithm that, to the best of our knowledge, is the first algorithm to deal with all of the following: limited continuous resources, uncertain action outcomes, and oversubscription planning. We tested HAO* in a realistic NASA simulation of a planetary rover, a complex domain of practical importance, and results show that it successfully handles problems with 2^{48} discrete states, as well as a continuous component. Its effectiveness comes from the fact that heuristic search can exploit resource constraints, as well as an admissible heuristic, in order to limit the reachable state space.

In our implementation, the HAO* algorithm is integrated with the dynamic programming algorithm of Feng et al. (2004) for solving hybrid MDPs. However HAO* can be integrated with other dynamic programming algorithms for solving hybrid MDPs. The Feng et al. algorithm find optimal policies under the limiting assumptions that transition probabilities are discrete, and rewards are either piecewise-constant or

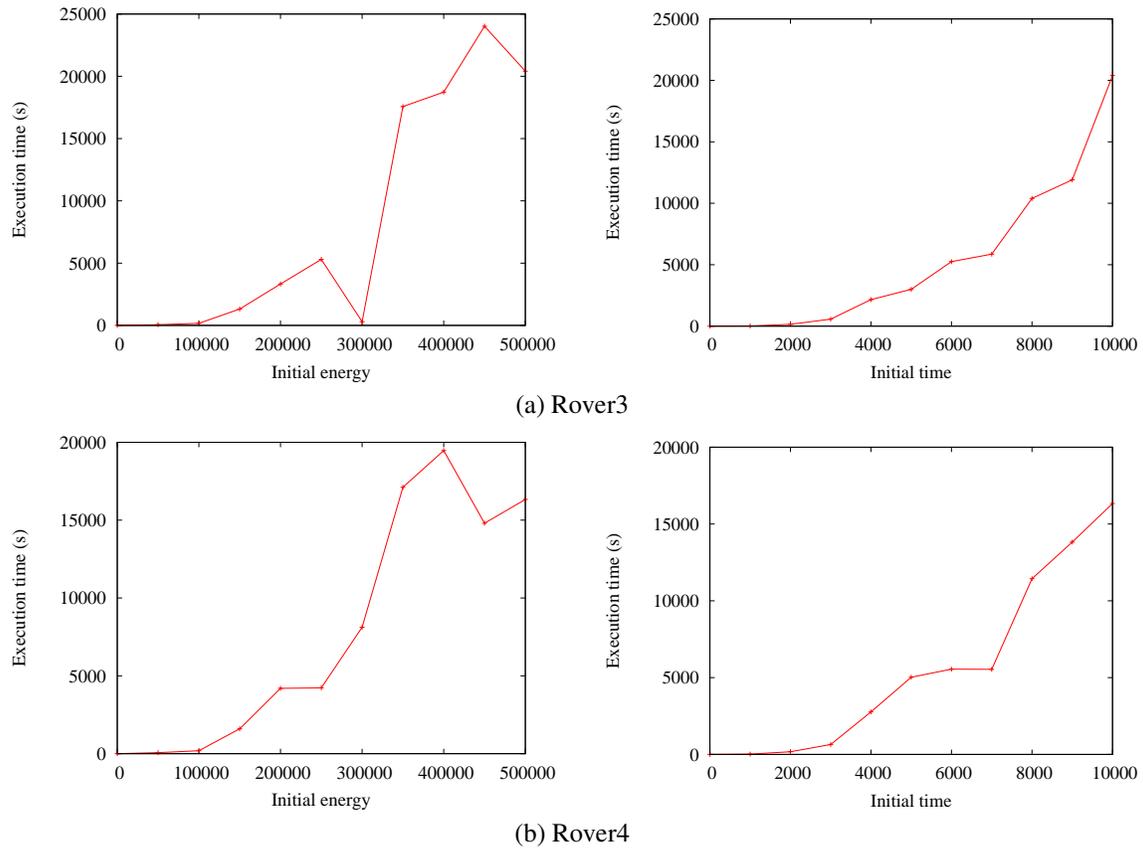


Figure 7: Same as Fig. 6 for the two largest benchmark problems.

piecewise-linear. More recently-developed dynamic programming algorithms make less restrictive assumptions, and also have the potential to improve computational efficiency (Li & Littman, 2005; Marecki, Koenig, & Tambe, 2007). Integrating HAO* with one of these algorithms could improve performance further.

There are several other interesting directions in which this work could be extended. In developing HAO*, we made the assumptions that every action consumes some resource and resources are non-replenishable. Without these assumptions, the same state could be revisited and an optimal plan could have loops as well as branches. Generalizing our approach to allow plans with loops, which seems necessary to handle replenishable resources, requires generalizing the heuristic search algorithm LAO* to solve hybrid MDPs (Hansen & Zilberstein, 2001). Another possible extension is to allow continuous action variables in addition to continuous state variables. Finally, our heuristic search approach could be combined with other approaches to improving scalability, such as hierarchical decomposition (Meuleau & Brafman, 2007). This would allow it to handle the even larger state spaces that result when the number of goals in an over-subscription planning problem is increased.

Acknowledgements

This work was funded by the NASA Intelligent Systems program, grant NRA2-38169. Eric Hansen was supported in part by NASA grant NAG-2-1463 and a NASA Summer Faculty Fellowship. This work was performed while Emmanuel Benazera was working at NASA Ames Research Center and Ronen Brafman was visiting NASA Ames Research Center, both as consultants for the Research Institute for Advanced Computer

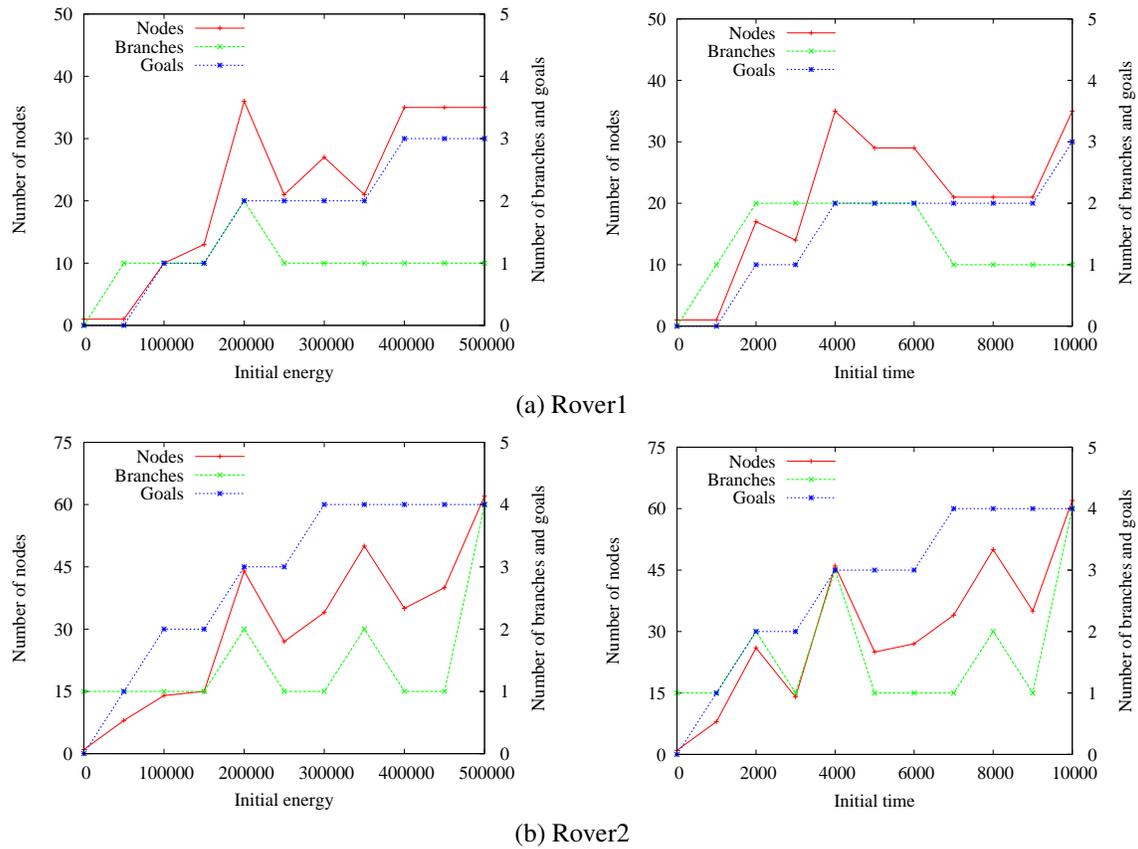


Figure 8: Complexity of the optimal policy: number of nodes, branches and goals in the optimal policy in the same setting as Fig. 4 to 7.

Science. Ronen Brafman was supported in part by the Lynn and William Frankel Center for Computer Science, the Paul Ivanier Center for Robotics and Production Management, and ISF grant #110707.

References

- Altman, E. (1999). *Constrained Markov Decision Processes*. Chapman and HALL/CRC.
- Asarin, E., & Schneider, G. (2002). Widening the boundary between decidable and undecidable hybrid systems. In *CONCUR'2002*, Vol. 2421 of *LNCS*, pp. 193–208, Brno, Czech Republic. Springer-Verlag.
- Bertsekas, D., & Tsitsiklis, J. (1996). *Neural Dynamic Programming*. Athena Scientific, Belmont, MA.
- Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11, 1–94.
- Boyan, J., & Littman, M. (2000). Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems 13*, pp. 1–7.

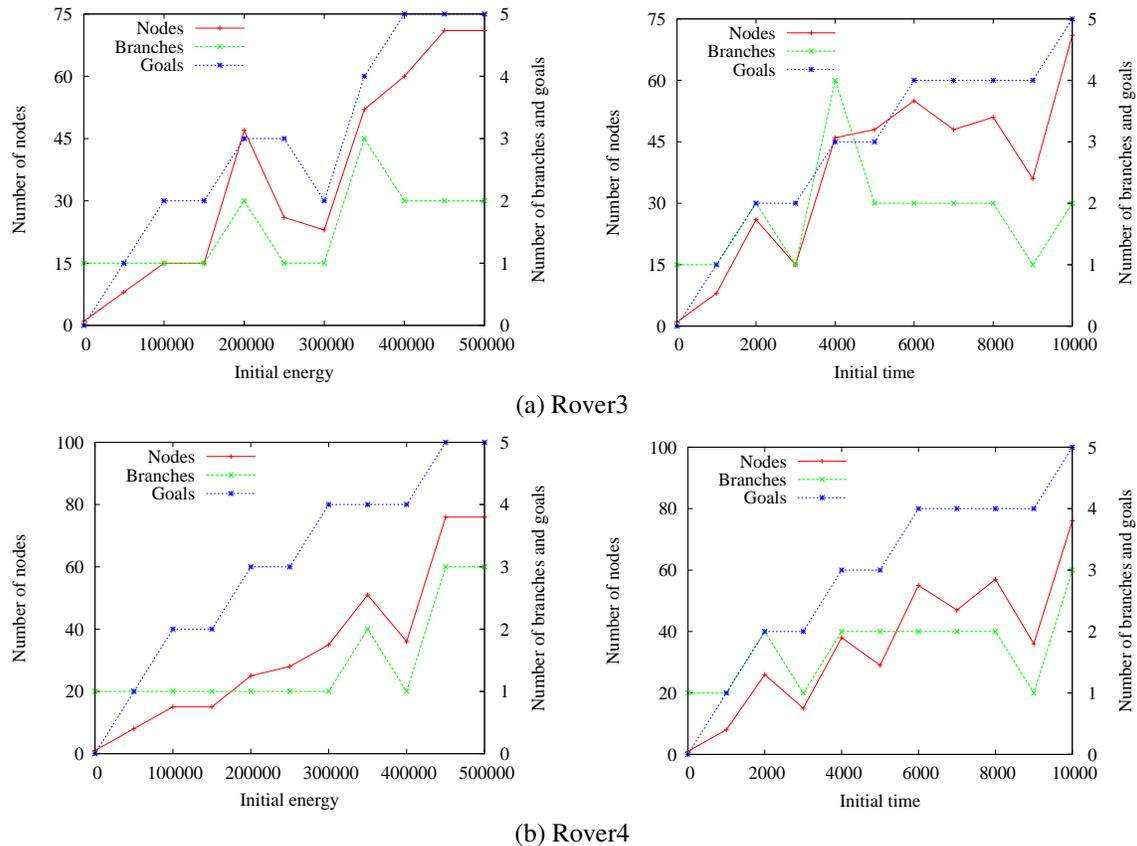


Figure 9: Same as Fig. 6 for the two largest benchmark problems.

- Bresina, J., Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D., & Washington, R. (2002). Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pp. 77–84.
- Bresina, J., Jonsson, A., Morris, P., & Rajan, K. (2005). Activity planning for the mars exploration rovers. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, pp. 40–49.
- Chakrabarti, P., Ghose, S., & DeSarkar, S. (1988). Admissibility of AO* when heuristics overestimate. *Artificial Intelligence*, 34, 97–113.
- Fehnker, A. (1998). Automotive control revisited: Linear inequalities as approximation of reachable sets.. In *HSCC*, pp. 110–125.
- Feng, Z., Dearden, R., Meuleau, N., & Washington, R. (2004). Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pp. 154–161.
- Friedman, J., Bentley, J., & Finkel, R. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Mathematical Software*, 3(3), 209–226.
- Hansen, E., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129, 35–62.

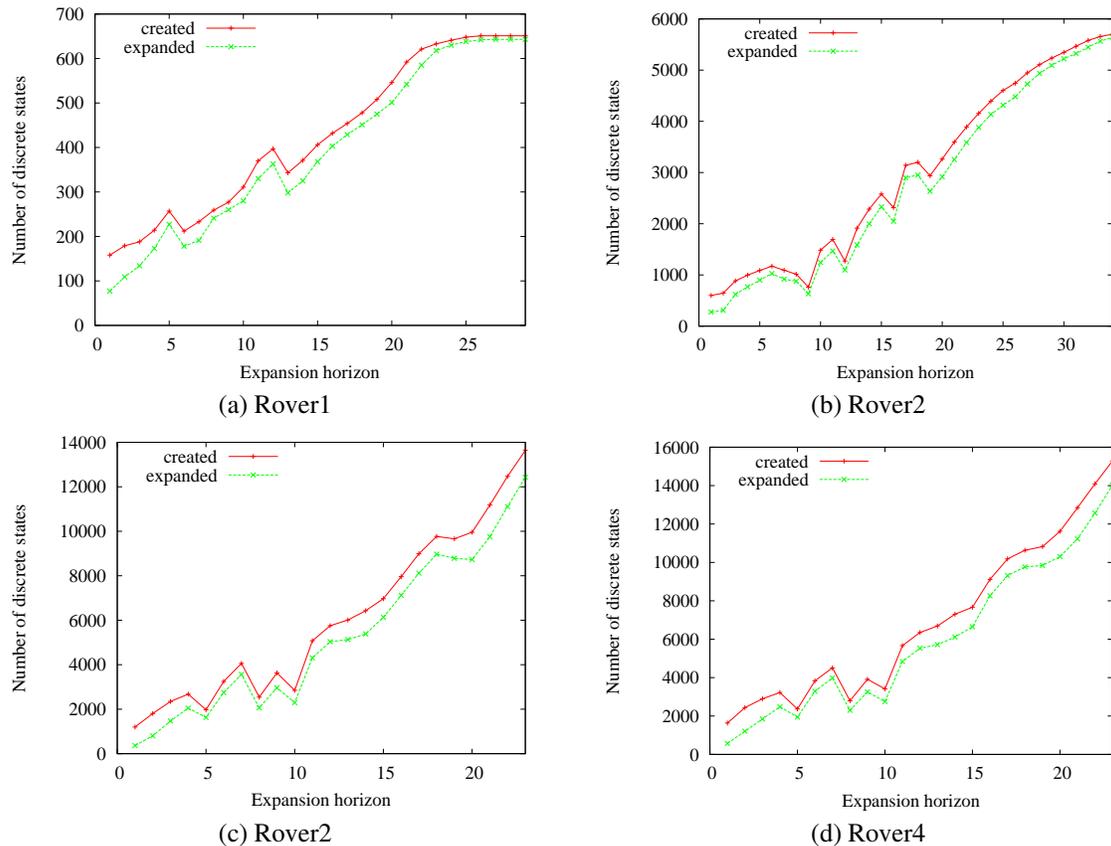


Figure 10: Influence of the expansion horizon on the number of nodes visited by the algorithm.

- Jimenez, P., & Torras, C. (2000). An efficient algorithm for searching implicit AND/OR graphs with cycles. *Artificial Intelligence*, 124, 1–30.
- Kveton, B., Hauskrecht, M., & Guestrin, C. (2006). Solving factored MDPs with hybrid state and action variables. *Journal of Artificial Intelligence Research*, 27, 153–201.
- Li, L., & Littman, M. (2005). Lazy approximation for solving continuous finite-horizon MDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pp. 1175–1180.
- Marecki, J., Koenig, S., & Tambe, M. (2007). A fast analytical algorithm for solving markov decision processes with real-valued resources. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 2536–2541.
- Mausam, Benazera, E., Brafman, R., Meuleau, N., & Hansen, E. (2005). Planning with continuous resources in stochastic domains. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 1244–1251. Professional Book Center, Denver, CO.
- Meuleau, N., & Brafman, R. (2007). Hierarchical heuristic forward search in stochastic domains. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*.
- Munos, R., & Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning*, 49(2-3), 291–323.

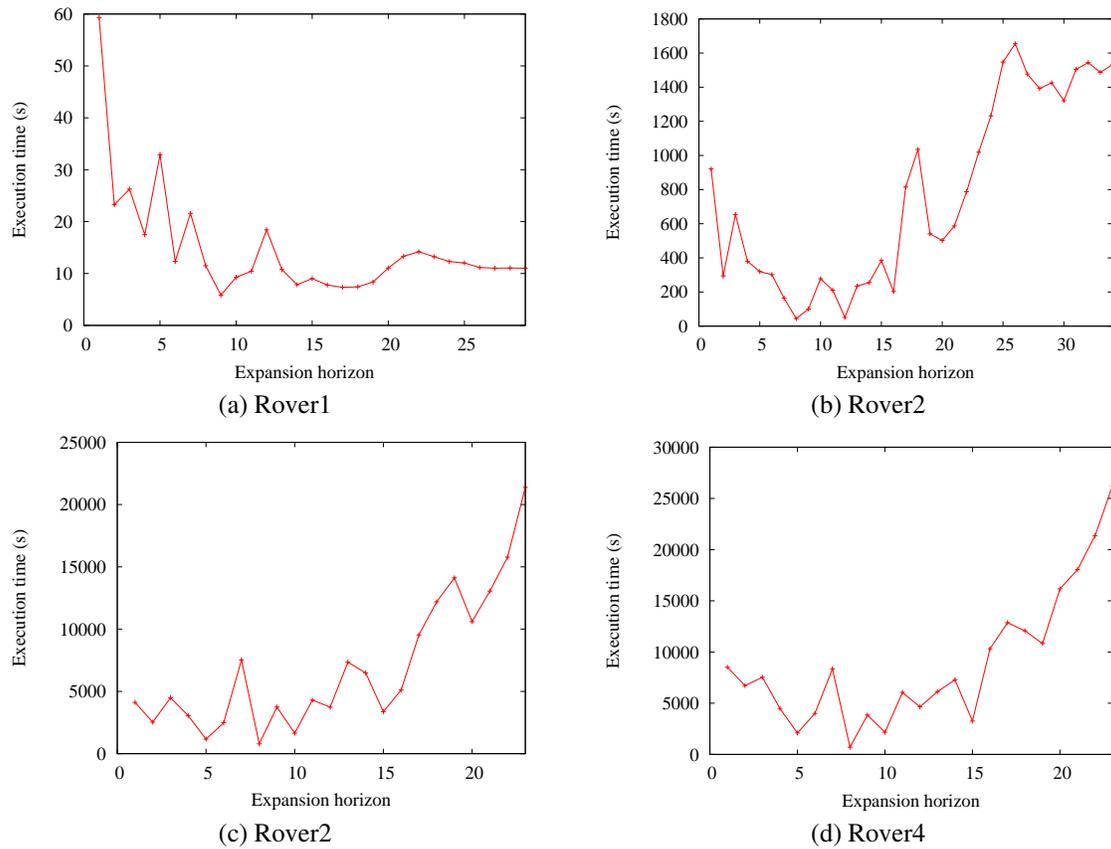


Figure 11: Influence of the expansion horizon on the execution time.

- Nilsson, N. (1980). *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, CA.
- Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.
- Pedersen, L., Smith, D., Deans, M., Sargent, R., Kunz, C., Lees, D., & Rajagopalan, S. (2005). Mission planning and target tracking for autonomous instrument placement. In *Proceedings of the 2005 IEEE Aerospace Conference*, Big Sky, Montana.
- Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, NY.
- Rust, J. (1997). Using randomization to break the curse of dimensionality. *Econometrica*, 65, 487–516.
- Smith, D. (2004). Choosing objectives in over-subscription planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pp. 393–401.
- Thiebaux, S., Gretton, C., Slaney, J., Price, D., & Kabanza, F. (2006). Decision-theoretic planning with non-markovian rewards. *Journal of Artificial Intelligence Research*, 25, 17–74.
- van den Briel, M., Sanchez, R., Do, M., & Kambhampati, S. (2004). Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pp. 562–569.