Planning with Continuous Resources in Stochastic Domains

Content Areas: Planning under uncertainty, Markov decision processes, Search.

Abstract

We consider the problem of optimal planning in stochastic domains with metric resource constraints. Our goal is to generate a policy whose expected sum of rewards is maximized for a given initial state. We consider a general formulation motivated by our application domain – planetary exploration – in which the choice of an action at each step may depend on the current resource levels. We adapt the forward search algorithm AO* to handle our continuous state space efficiently.

1 Introduction

There are many problems inherent in communication with remote devices such as planet exploratory rovers [Bresina *et al.*, 2002]. Therefore, remote rovers must operate autonomously over substantial periods of time. Moreover, the surfaces of planets are very uncertain environments: there is a great deal of uncertainty in the duration, energy consumption, and outcome of a rover's actions. Currently, instructions sent to planetary rovers are in the form of a simple plan for attaining a single goal (e.g., photographing some interesting rock). The rover attempts to carry this out, and when done remains idle. If it fails early on, it makes no attempt to recover and possibly achieve an alternative goal. This may have serious impact on missions. For instance, it has been estimated that the 1997 Mars Pathfinder rover spent between 40% and 75% of its time doing nothing because plans did not execute as expected.

Working in this application domain, our goal is to provide a planning algorithm that can generate a reliable contingent plan that can respond to different events and action outcomes. This plan must optimize the expected value of the experiments conducted by the rover, while being aware of its time, energy, and memory constraints. In particular, we must pay attention to the fact that given any initial state, there are many experiments the rover could conduct, *most combinations of which* are infeasible due to resource constraints. General features of our problem include: (1) concrete starting state; (2) continuous resources (including time) with stochastic consumption; (3) uncertain action effects; (4) several possible one-time-rewards, only a subset of which are achievable. This type of problem is of general interest, as it fits a large class of (stochastic) logistics problems, and many more.

Past work has dealt with various variants of this problem. Related work on MDPs with resource constraints includes the model of constrained MDPs developed in the OR community [Altman, 1999]. In this model, a linear program includes constraints on resource consumption and is used to find the best feasible policy, given an initial state and resource allocation. But a drawback of the constrained MDP model is that it does not include resources in the state space, and thus a policy cannot be conditioned on resource availability. Moreover, resource consumption is modeled as deterministic. In the area of decision-theoretic planning, several techniques have been proposed to handle uncertain continuous variables (e.g. [Feng *et al.*, 2004; Younes and Simmons, 2004]). Finally, [Smith, 2004; van den Briel *et al.*, 2004] considered the problem of over-subscription planning, i.e., planning with a large set of goals which is not entirely achievable. They provide techniques for selecting a subset of goals for which to plan, but they deal only with deterministic domains.

Our main contribution is an implemented algorithm that handles all of these problems together: oversubscription planning, uncertainty, and limited continuous resources. Our approach is to include resources in the state description. This allows decisions to be made based on resource availability, and it also allows resource consumption to be stochastic (which contrast with the factored MDP approach). Although this increases the size of the state space, we assume that the value functions may be represented compactly and we use the work of Feng et al. (2004) on piecewise constant and linear approximations of dynamic programming (DP) in our implementation. However, DP cannot use the resource constraint to reduce the search space of feasible policies because it solves the problem for each state without considering the trajectory along which the state was reached Our contribution in this paper is to show how to use the forward heuristic search algorithm called AO* [Pearl, 1984; Hansen and Zilberstein, 2001] to solve MDPs with resource constraints and continuous resource variables. Unlike DP, forward search keeps track of the trajectory from the start state to each reachable state, and thus it can check whether the trajectory is feasible or violates a resource constraint. This allows heuristic search to prune infeasible trajectories and can dramatically reduce the number of states that must be considered to find an optimal policy. This is particularly important in our domain where the discrete state space is huge (exponential in the number of goals), yet the portion reachable from any initial state is relatively small because of the resource constraints. It is well-known that heuristic search can be more efficient than DP because it leverages a search heuristic and reachability constraints to focus computation on the relevant parts of the state space. We show that for problems with resource constraints, this advantage can be even greater than usual because resource constraints further limit reachability.

2 Problem definition and solution approach

Problem definition We consider a Markov decision process (MDP) with both continuous and discrete state variables. Continuous variables typically represent resources, where one possible type of resource is time. Discrete variables model other aspects of the state, including (in our application) the set of goals achieved so far by the rover. (Keeping track of already-achieved goals ensures a Markovian reward structure, since we reward achievement of a goal only if it was not achieved in the past.) Although our models typically contain multiple discrete variables, this plays no role in the description of our algorithm, and so, for notational convenience, we model the discrete component as a single variable.

A *Markov state* $s \in S$ is a pair (n, x) where $n \in N$ is the discrete variable, and $\mathrm{x} = (x_i)$ is a vector of continuous variables. For each $x_i \in X_i$, X_i is an interval of the real line, and $\mathbf{X} = \bigotimes_i X_i$ is the hypercube over which the continuous variables are defined. We assume an explicit *initial state*, denoted (n_0, x_0), and one or more absorbing *terminal states*. One terminal state corresponds to the situation in which all goals have been achieved. Others model situations in which resources have been exhausted or an action has resulted in some error condition that requires executing a safe sequence by the rover and terminating plan execution.

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

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

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

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

In our application domain, continuous variables model unreplenishable resources. We also assume that each action has some minimal positive consumption of at least one resource. An important implication of this assumption is that the number of possible steps in any execution of a plan is bounded, which we refer to by saying the problem has a *bounded horizon*. Note that the actual number of steps until termination can vary depending on actual resource consumption.

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

$$\begin{aligned}
V_n^0(\mathrm{x}) &= 0 \ , \\
V_n^{t+1}(\mathrm{x}) &= \max_{a \in A_n(\mathrm{x})} \left[Q_{n,a}^{t+1}(\mathrm{x}) \right] \ , \\
Q_{n,a}^{t+1}(\mathrm{x}) &= \sum_{n' \in N} \Pr(n' \mid n, \mathrm{x}, a) \qquad (1) \\
&\int_{\mathrm{x}'} \Pr(\mathrm{x}' \mid n, \mathrm{x}, a, n') \left(R_{n'}(\mathrm{x}') + V_{n'}^t(\mathrm{x}') \right) d\mathrm{x}'
\end{aligned}$$

where $A_n(\mathrm{x})$ denotes the set of actions executable in (n, x). Note that the index t represents sequential order but does not necessarily correspond to time in the planning problem. The duration of actions is one of the biggest source of uncertainty in our rover problems, and we typically model time as one of the continuous resources x_i.

Solution approach Feng et al. [Feng *et al.*, 2004] describe a dynamic programming (DP) algorithm that solves this Bellman optimality equation. In particular, they show that the continuous integral over x' can be computed exactly, as long as the transition function satisfies certain conditions. We defer a discussion of the details of their approach until Section 3.3, and treat this computation as a black-box for now. This allows us to simplify the description of our algorithm in the next section and focus on our contribution.

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

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

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

We note that LAO*, a generalization of AO*, allows for policies that contain "loops" in order to specify behavior over an infinite horizon [Hansen and Zilberstein, 2001]. Because our assumptions about resource consumption imply that our problem has a bounded horizon, AO* suffices. However, similar ideas can be used to extend LAO* to our setting.

3 The Algorithm

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

• The *explicit graph* describes all the states that have been expanded so far and the AND/OR edges that connect them. The nodes of the explicit graph are stored in two lists: OPEN and CLOSED.

• The *greedy policy* (or partial solution) graph is a sub-graph of the explicit graph describing the current optimal policy.

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

3.1 Data Structures

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

• The value of the discrete state. In our application these are the discrete state variables and set of goals achieved.

• Pointer to its parents and children in the explicit and greedy policy graphs, as pairs (n', a), where n' is a parent/child node, and a is an action that allows this transition.

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

• $H_n(\cdot)$ – the heuristic function. For each $\mathbf{x} \in \mathbf{X}$, $H_n(\mathbf{x})$ is a heuristic estimate of the optimal expected reward from state $(n, \mathbf{x})$. The heuristic functions H are obtained by solving a relaxed problem. An admissible heuristic is obtained by assuming that all action consumptions take their smallest possible value in each dimension with probability 1.

• $V_n(\cdot)$ – the value function. At the leaf nodes of the explicit graph, $V_n = H_n$. At the non-leaf nodes of the explicit graph, V_n is obtained by backing up the H functions from the descendant leaves. If the heuristic function $H_{n'}$ is admissible in all leaf nodes n', then $V_n(\mathbf{x})$ is an upper bound on the optimal reward to come from $(n, \mathbf{x})$ for all $\mathbf{x}$ reachable under the greedy policy.

• g_n – a heuristic estimate of the increase in value of the greedy policy that we would get by expanding node n. If $H_{n'}$ is admissible then g_n represents an upper bound on the gain in expected reward. The gain g_n is used to determine the priority of nodes in the OPEN list ($g_n = 0$ if n is in CLOSED), and to bound the error of the greedy solution at each iteration of the algorithm.[1]

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

3.2 Algorithm

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

Expanding a node (lines 10 to 20): At each iteration, the algorithm expands the open node n with the highest priority g_n in the greedy graph. Note that standard AO* expands only tip nodes, whereas this algorithm can put a node back in the OPEN list, that has been expanded earlier and that belongs to the greedy policy (lines 18 & 23). The algorithm then considers all possible successors (a, n') of n given the state distribution P_n. Typically, when n is expanded for the first time, we enumerate all actions a possible in $(n, \mathbf{x})$ ($a \in A_n(\mathbf{x})$) for some reachable $\mathbf{x}$ ($P_n(\mathbf{x}) > 0$), and all arrival states n' that can result from such a transition ($\Pr(n' \mid n, \mathbf{x}, a) > 0$).[2] If n' was previously expanded (thus it has been put back in OPEN), only actions and arrival nodes not yet expanded are considered. In line 11, we check whether a node has already been generated. This is not necessary if the graph is a tree (i.e., there is only one way to get to each discrete state).[3] In

[1]The algorithm keeps its convergence properties if we use a *heuristic* other than g to select the next node to expand. However it loses its anytime properties.

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

[3]Sometimes it is beneficial to use the tree implementation of AO*

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

Algorithm 1: AO* algorithm for hybrid domains.

line 15, a node n' is terminal if $M(P_{n'}) = 0$, i.e. if we run out of a resource before arriving in n'. In our application domain each goal pays only once, thus the nodes in which all goals of the problem have been achieved are also terminal. Finally, the test in line 19 prevents loops in the explicit graph, as discussed in section 2.

Putting a node from CLOSED back in OPEN when it is regenerated is not a feature of standard AO* as described in [Hansen and Zilberstein, 2001]. As a search node represents several problem states, when a new path to an existing node is found, it may have reached some Markov states that were not considered in the explicit graph before, and so it needs to be expanded. For this reason, when a new path to n' is found, the state distribution in $P_{n'}$ may need to be updated and actions that where not possible in n' before may become applicable. Consequently new arrival nodes may also become possible. Therefore, n' needs to be expanded again.

Updating the value functions (lines 22 to 23): As in standard AO*, the value of a newly expanded node must be updated. This consists of recomputing its value function with Bellman's equations (Eqn. 1), based on the value functions of all children of n in the explicit graph. This computation is discussed in Section 3.3. Note that these backups involve all continuous states $\mathbf{x} \in \mathbf{X}$, *not* just values. However, they consider only actions and arrival nodes that are reachable according to P_n. Once the value of a state is updated, its new value must be propagated backward in the explicit graph. The backward propagation stops at nodes where the value function is not modified, and/or at the rood node. The whole process is performed by applying Algorithm 2 to the newly expanded node.

when the problem graph is *almost* a tree, by duplicating nodes that represents the same (projected) state reached through different paths.

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

Algorithm 2: Updating the value functions V_n.

Updating the state distributions (line 23): P_n's represent the state distribution *under the greedy policy*, and they need to be updated after recomputing the greedy policy. More precisely, P needs to be updated in each descendant of a node where the optimal decision changed. To update a node n, we consider all its parents n' in the greedy policy graph, and all the actions a that can lead from one of the parents to n. The probability of getting to n is the sum over all (n', a) of the probability of arriving from n' under a, which is obtained by convoluting $P_{n'}$ and the transition probability of a:

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

Note that it is sufficient to consider only pairs (n', a) where a is the greedy action in n' for some reachable resource level:

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

where $\mu^*_n(\mathbf{x}) \in A$ is the greedy action in $(n, \mathbf{x})$. Note that this operation may induce a loss of total probability mass ($P_n < \sum_{n'} P_{n'}$) because we can run out of a resource during the transition and end up in a sink state. When the distribution P_n of a node n in the OPEN list is updated, its priority g_n is recomputed using the following equation (the priority of nodes in CLOSED is maintained as 0):

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

where $S(P)$ is the support of P: $S(P) = \{\mathbf{x} \in \mathbf{X} : P(\mathbf{x}) > 0\}$, and X^{old}_n contains all $\mathbf{x} \in \mathbf{X}$ such that the state $(n, \mathbf{x})$ has already been expanded before ($X^{\text{old}}_n = \emptyset$ if n has never been expanded). The techniques used to represent the continuous probability distributions P_n and compute the continuous integrals are discussed in Section 3.3. Algorithm 3 presents the state distributions updates. It applies to the set of nodes where the greedy decision changed during value updates (including the newly expanded node, i.e. n in Algorithm 1).

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

Algorithm 3: Updating the state distributions P_n.

3.3 Handling Continuous Variables

Computationally, the most challenging aspect of the algorithm is the handling of continuous state variables, and particularly the computation of the continuous integral in Bellman backups. We approach this problem using the ideas developed in [Feng *et al.*, 2004] for the same application domain. However, the presented algorithm remains identical for other models of the uncertainty on continuous variables, as long as value functions can be computed in finite time. The basic idea is to exploit the apparent structure in the continuous value functions of the type of problems we are addressing. These value functions typically appear as collections of humps and plateaus, each of which corresponds to a region in the state space where similar goals are pursued by the optimal policy. Such structure is exploited by grouping states that belong to the same plateau, while reserving a fine discretization for the regions of the state space where it is the most useful (such as the edges of plateaus).

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

3.4 Properties

As for standard AO* [Hansen and Zilberstein, 2001], it can be showed that if the heuristic functions H_n are admissible (optimistic), and *if the continuous back-ups are computed exactly*, then: (i) at each step of the algorithm, $V_n(\mathrm{x})$ is an upper-bound on the optimal expected return in (n, x), for all (n, x) expanded by the algorithm; (ii) the algorithm terminates after a finite number of iterations; (iii) after termination, $V_n(\mathrm{x})$ is equal to the optimal expected return in (n, x), for all (n, x) reachable under the greedy policy ($P_n(\mathrm{x}) > 0$). Moreover, if we assume that, in each state, there is a *done* action that terminates execution with zero reward then we can evaluate the greedy policy at each step of the algorithm by assuming that execution is ends each time we reach a leaf of the greedy subgraph. Under the same hypotheses, the error of the greedy policy at each step of the algorithm is bounded by $\sum_{n \in \text{GREEDY} \cap \text{OPEN}} g_n$. This property allows trading computation time for accuracy by stopping the algorithm early.

4 Experimental Evaluation

We tested our algorithm on a slightly simplified variant of the rover model used for NASA Ames October 2004 IS demo [Pedersen *et al.*, 2005]. In this domain, a planetary rover moves in a planar graph made of locations and paths, sets up instruments at different rocks, and performs experiments on the rocks. Actions may fail, and their energy and time consumption are uncertain. The problem instance used in our preliminary experiments contains 43 propositional state variables, 37 actions and 5 goals (rocks to be tested). Therefore, there are 2^{48} different discrete states, which is far beyond the reach of a flat DP algorithm. Resource consumptions are drawn from two type of distributions: uniform and normal, and then discretized. The results presented here were obtained using a preliminary implementation of thepiecewise constant DP approximations described in [Feng *et al.*, 2004] and using a flat representation of state partitions instead of kd-trees. This is considerably slower than an optimal implementation. To compensate, our domain features a single abstract continuous resource, while the original domain contains two resources (time and energy). We used the following admissible heuristic: H_n is the constant function equal to the sum of the utilities of all the goals not achieved in n.

We varied the initial amount of resource available to the rover. As available resource increases, more nodes are reachable and more reward can be gained. The performance of the algorithm is presented in Table 1. We see that the number of reachable discrete states is much smaller than the total number of states (2^{48}) and the number of nodes in an optimal policy is surprisingly small. This indicates that AO* is particularly well suited to our rover problems. However, the

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

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

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

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

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

number of nodes expanded is quite close to the number of reachable discrete states. Thus, our current simple heuristics is only slightly effective in reducing the search space, and reachability makes the largest difference. This suggests that much progress can be obtained by using better heuristics. The last column measures the total number of reachable Markov states, after discretizing the action consumptions as in [Feng *et al.*, 2004]. This is the space that a forward search algorithm manipulating Markov states, instead of discrete states, would have to tackle. In most cases, it would be impossible to explore such space with poor quality heuristics such as ours. These numbers indicate that our algorithm is quite effective in scaling up to very large problems by exploiting the structure presented by continuous resources.

When g_n is admissible, we can bound the error of the current greedy graph by summing its value over fringe nodes. In Table2 we describe the time/value tradeoff we found for this domain. On the one hand, we see that even a large compromise in quality leads to no more than 25% reduction in time. On the other hand, we see that much of this reduction is obtained with a very small price ($\epsilon = 0.5$). Additional experiments are required to learn if this is a general phenomenon.

5 Conclusions

We presented a variant of the AO* algorithm that, to the best of our knowledge, is the first algorithm to deal with: limited continuous resources, uncertainty, and oversubscription planning. Our preliminary implementation of this algorithm shows very promising results on a domain of practical importance. We are able to handle problem with 2^{48} discrete state, as well as a continuous component.

We are now implementing the full algorithm, on whose performance we shall report in the final version. This algorithm includes: (1) a full implementation of the techniques described in [Feng *et al.*, 2004]; (2) a rover model with two continuous variables; (3) a more informed heuristic function. We will generate this heuristic function by solving the original planning problem while assuming deterministic transitions for the continuous variables, i.e., $Pr(\mathbf{x}'|n, \mathbf{x}, a, n') \in \{0, 1\}$. If we assume actions consumes the minimal amount of each resource, we obtain an admissible heuristic function. A (probably) more informative, but inadmissible heuristic function is obtained by using the mean resource consumption. Our central idea is to use the *same algorithm* to solve both the relaxed and original problem and to use the value function V_n for the relaxed problem as the heuristic function. The relaxed problem is easier to solve, and unlike typical heuristic functions which are recomputed for each search state, one expansion from the initial state provides us with values that can be used for most reachable nodes.

References

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

[Bresina *et al.*, 2002] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. of UAI-02*, pages 77–84, 2002.

[Feng *et al.*, 2004] Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous Markov decision problems. In *Proc. of UAI-04*, pages 154–161, 2004.

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

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

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

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

[Smith, 2004] D. Smith. Choosing objectives in over-subscription planning. In *Proc. of ICAPS-04*, pages 393–401, 2004.

[van den Briel *et al.*, 2004] M. van den Briel, M.B. Do R. Sanchez and, and S. Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proc. of AAAI-04*, pages 562–569, 2004.

[Younes and Simmons, 2004] H.L.S. Younes and R.G. Simmons. Solving generalized semi-Markov decision processes using continuous phase-type distributions. In *Proc. of AAAI-04*, pages 742–747, 2004.