Stochastic Over-subscription Planning using Hierarchies of MDPs

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

Abstract

In over-subscription planning (OSP), the set of goals is not achievable jointly, and the task is to find a plan that attains the best feasible subset of goals given resource constraints. Recent classical OSP algorithms ignore the uncertainty inherent in many natural application domains where OSPs arise. And while modeling stochastic OSP problems as MDPs is easy, the resulting models are too large for standard solution approaches. Fortunately OSP problems have a natural two-tiered hierarchy, and in this paper we adapt and extend tools developed in the hierarchical reinforcement learning community in order to effectively exploit this hierarchy and obtain compact, factored policies. Typically, such policies are sub-optimal, but under certain assumptions that hold in our planetary exploration domain, our factored solution is, in fact, optimal. Our algorithms work by repeatedly solving a number of smaller MDPs, while propagating information between them. We evaluate a number of variants of this approach on a set of stochastic instances of a planetary rover domain, showing substantial performance gains.

Introduction

Over-subscription planning problems (OSPs)[1] (Smith 2004) are classical planning problems in which the given set of goals is not achievable jointly. The task is to find a plan that achieves some optimal feasible subset of the goal set given resource constraints. OSP is a natural generalization of classical planning problems with their fixed goal set, and they arise in numerous important application domains. Interesting and useful examples of OSPs abound, including service scheduling, logistics problem with constrained resources, and more generally, problems that involve multiple sub-tasks constrained by shared resources. This paper is motivated by our work on the planetary exploration domain, and in particular, our desire to scale-up algorithms for planetary rover problems.

There are different formulations of this problem, some in which goals have rewards and actions have costs, such as (Sanchez and Kambhampati 2005), and some in which qualitative metrics are used to assess different sub-goals (Brafman and Chernyavsky 2005). This work adopts a decision theoretic approach where the goal is to maximize the (expected) utility of the goals achieved under the constraints imposed by limited resources.

Many application domains in which OSPs are used, and planetary exploration in particular, exhibit large amounts of uncertainty. For instance, the effects of the actions performed by planetary rovers are stochastic, with resource consumption, distance travelled, etc., being affected by variable unpredictable features of the environment, such as weather and terrain. It is thus natural to consider stochastic OSPs (SOSPs), i.e., OSPs in which the effects of actions are stochastic. Such OSPs are modelled naturally as Markov Decision Processes (MDPs) (Puterman 1994). MDPs naturally capture both the stochastic effect of actions as well as the ability of multiple events/states to provide value to the agent. However, the state-space of the resulting MDP grows exponentially with the number of possible goals in the problem. These goals are not independent, often indirectly linked via their consumption or use of shared resources, such as energy, time, or shared means of transportation. Optimally solving such MDPs within a reasonable time using current methods is infeasible given a large set of possible sub-goals.

As observed by Smith (2004), OSPs lend themselves to a natural two-level hierarchical model. At the top-level, the problem is one of scheduling the different sub-tasks. For instance, in planetary rover exploration, decisions at this level involve the choice of the next location to visit, the rocks to track or stop tracking, and possibly the instruments to warm-up. The sub-tasks appear at the lower level of this hierarchy, e.g., performing experiments and collecting data at a particular site. The main contribution of this paper is an approach to solving stochastic OSPs that exploits this hierarchy and repeatedly solves substantially smaller MDPs that describe each sub-task. As our experimental results indicate, as the sub-tasks increase in complexity and as their number increases, our algorithm becomes considerably faster than similar algorithms that solve the flat representation of the domains. Moreover, we show that under a certain assumption that applies to the type of domains that motivate this work – the solution we obtain is both compact and optimal.

Our work is closely related to work on hierarchical reinforcement learning (Dietterich 2000; Andre and Russell

[*] QSS Group Inc.

[†] Research Institute for Advanced Computer Science.

[1] We use "OSP" both as an abbreviation of "over-subscription planning", and "over-subscription planning problems".

2002). We farther develop solution techniques that were discussed in the past, e.g., in (Andre 2003). This line of work concentrated on learning both because domain models are not always available, and also because model-based methods must compute transition functions for macro operators (see below) – a task considered computationally expensive (Andre 2003). Indeed, we have found naive macro computation extremely expensive, and one of our contributions is to show that with more clever methods and with particular problem structure, macros can be computed quite fast. Unlike most past work, our techniques are also able to generate globally optimal policies under certain assumptions. Overall, the approach we present is more unified in its treatment of abstraction and hierarchy – the two are intertwined, features a new factored variant of the off-line policy iteration algorithm as opposed to the on-line algorithms used in Hierarchical RL, and offers fast macro computation methods. We evaluate these algorithms on a toy model of the planetary rover domain.

Problem Formulation

We briefly review the Markov Decision Process (MDP) model. We then explain how we recast stochastic OSPs as a special class of MDPs.

Factored MDPs

A Markov Decision Process (MDP) is a four-tuple $\langle S, A, T, R\rangle$, where S is a set of states, A is a set of actions, $T : S \times A \times S \rightarrow [0, 1]$ is the transition function which specifies for every two states $s, s' \in S$ and action $a \in A$ the probability of making a transition from s to s' when a is executed, and $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function.

We are interested in factored MDPs which have the form $\langle X, A, T, R\rangle$. Here X is a set of state variables, and A, T, R, are as before. The variables in X induce a state space, consisting of the Cartesian product of their domains. Typically, it is assumed that the transition function T is also described in a compact manner that utilizes the special form of the state space, such as dynamic Bayes net (Dean and Kanazawa 1993) or probabilistic STRIPS rules (Hanks and McDermott 1994). In this paper, we do not commit to any particular action description. However, we implicitly assume that it is easy to identify the relevant variables with respect to an action $a \in A$. This is the set of variables whose value can change when a is executed, as well as those variables that affect the probability by which these variables change their value. We use $inf(a)$ to denote this set of variables.

Finally, we are interested in problems with a concrete initial state. Thus, we slightly modify the definition of an MDP to $\langle X, A, T, R, I\rangle$, where I is a concrete initial state, i.e., an assignment of value to each variable in X.

Stochastic OSPs The term *oversubscription planning* refers to classical planning problems in which we have a set of sub-tasks, or sub-goals, that cannot be achieved jointly because of resource limitation. A weight, or value, is associated with each sub-goal, and the task is to generate a plan that achieves the maximal feasible set of sub-goals with respect to total weight. In this work, we assume that different goals have different utility and we aim at maximizing the expected utility of the plan.

The loose coupling of goals/sub-tasks plays an important role in our approach, as well as in other approaches to OSPs such as (Smith 2004). It is assumed that sub-tasks are pretty much independent, coupled only through their effect on a number of shared state variables which includes, but is not limited to, the limited resources (such as time, energy, memory, etc.). Aside from their effect on shared variables, actions affect only variables local to the task. The fact that subtasks are localized makes it easy to decompose OSPs. Indeed, domain decomposition methods, and in particular, the general method proposed by (Amir and Engelhardt 2003) naturally leads to a 2-tiered hierarchical model of OSPs. The leaf nodes in this hierarchy describe the local variables and actions for each particular task, as well as those global parameters required to perform them. The root node describes the global parameters and is in charge mostly, but not only, of selecting the next task to perform. The root and each of its children share variables, but each sub-process has its own actions. We assume such a decomposition as part of *the input* of our problem. We refer the reader to (Amir and Engelhardt 2003) for more details on how such decompositions can be formed automatically in classical planning problems. These ideas (but not the associated planning algorithms) extend naturally to MDPs, and they motivated our formulation of the problem.

A *Stochastic Over-subscription Planning problem* (SOSP) is a set $\{M_0, \ldots, M_n\}$ of factored MDPs, where

- $M_i = \langle X_i, A_i, T_i, R_i, I_i\rangle$.
- $X_i \bigcap X_j \subseteq X_0$ for all $i > 0, j > 0$ such that $i \neq j$. This is called the *running intersection* property.
- $A_i \bigcap A_j = \emptyset$ for all $i \neq j$.
- Each $X_i : i > 0$ contains a special boolean fluent $Done_i$ such that $Done_i \in X_0$ and $Done_i \notin X_j$ for $j > 0, j \neq i$.
- R_i is 0 everywhere, except for triples of the form (s, a, s') such that $Done_i$ is true in s' and not in s.
- For every $i > 0$, no action can change $Done_i$ from *true* to *false*.
- The initial states I_i agree on shared variables.

The requirement that actions belong to a single sub-process and that initial states agree on shared variables implies that $M = \langle X, A, T, R, I\rangle$, where $X = \bigcup X_i$, $A = \bigcup A_i$, $R = \sum R_i$, and $I = \bigcup I_i$ is an MDP in which for every $a_i \in A_i$ we have that $inf(a_i) \subseteq X_i$. Thus, a SOSP is simply an MDP with a special structure. The fact that rewards are only obtained when $Done_i$ becomes true, and that $Done_i$ cannot become false, gives us the type of one-time reward for accomplishing some sub-task associated with OSPs. Indeed, the structure of SOSPs is closely related, and generalizes, the structure reflected in the Orienteering Problem representation of (Smith 2004). And it is this special structure – together with one additional assumption discussed later – that we wish to exploit. Note, however, that because X_0 contains one $Done_i$ fluent for every $i > 0$, the state space

of M_0 remains exponential in n, but it is still much smaller than the original space with its $O(2^{\sum_i |X_i|})$.

The sub-process M_0 is called the *root* process. The sub-processes $M_1, \ldots, M_n$ are called the child (or leaf) processes. For each set of process variables X_i, we define:

- $\bar{X}_i = X_i \cap X_0$ are the separator (shared) variables between M_0 and M_i.
- $\widetilde{X}_i = X_i - X_0$ are the private variables of M_i.
- $X_{0-i} = X_0 - X_i$ is the difference of M_0 and M_i.
- The set of private variables of M_0 is defined as $\widetilde{X}_0 = X_0 - \bigcup_{i=1}^{n} X_i = \bigcap_{i=1}^{n} X_{0-i}$.

As we noted above, the state space of factored MDPs is the Cartesian product of their domains. Thus, we can naturally define various classes of states (where we assumed Boolean variables to simplify notation): $S = 2^X$, $S_i = 2^{X_i}$, $\bar{S}_i = 2^{\bar{X}_i}$, $\widetilde{S}_i = 2^{\widetilde{X}_i}$, $S_{0-i} = 2^{X_{0-i}}$. Since, the sets $\tilde{X}_i$ ($i \geq 0$) and $\bar{X}_i$ ($i \geq 1$) constitute a partition of X, each Markov state $s \in S$ can be decomposed in various ways, such as $s = (\tilde{s}_0, \bar{s}_1, \tilde{s}_1, \ldots \bar{s}_n, \tilde{s}_n)$, $s = (s_0, \tilde{s}_1, \ldots \tilde{s}_n)$ and $s = (\tilde{s}_0, s_1, \ldots s_n)$. Similarly, we will use the notations $s_0 = (s_{0-i}, \bar{s}_i) \in S_0 = S_{0-i} \times \bar{S}_i$ and $s_i = (\bar{s}_i, \tilde{s}_i) \in S_i = \bar{S}_i \times \widetilde{S}_i$.

Policies

So far, we have not explicitly said what we want to do with our special MDP. Typically, given an MDP, one seeks a policy, i.e., a mapping from states to actions, that maximizes some function of the reward stream. We would like to maximize the expected sum of rewards – this criteria is well defined because we can at most get rewarded n times in our model. However, we want to obtain this maximal reward using a compact policy that reflects the structure of the problem. Intuitively, this policy would start with M_0's policy which basically decides which sub-goal to achieve next, and then for each sub-goal, we would follow the policy of the corresponding M_i.

In the simplest case, we define a factored policy to be the sequence $\langle \mu_0, \mu_1 \ldots, \mu_n \rangle$, where $\mu_0 : S_0 \rightarrow A_0 \cup \{\mu_1, \ldots, \mu_n\}$; and $\mu_i : S_i \rightarrow A_i \cup \{Abort\}$, $i > 0$. This is to be understood as follows: the root process can either execute a local primitive action $a_0 \in A_0$, or call a sub-process through the *macro-action* μ_i. Each local policy μ_i specifies, in each sub-process state, a choice between an action private to that process, and passing control back to the root process through the $Abort$ action. In this work, we consider the case where the root process may use several different macro-actions μ_i for each sub-process M_i. More precisely, the root process has a different macro-action $\mu_i[s_{0-i}] : S_i \rightarrow A_i \cup \{Abort\}$ for each vector $s_{0-i} \in S_{0-i}$, and $\mu_i[s_{0-i}]$ may be used only in states $s_0 = (s_{0-i}, \bar{s}_i) \in S_0$. The reasons for this choice will become apparent in the following.

A factored policy uses only part of the state space to make action choices. For instances, choices of actions in S_0 depend only on the root process variables X_0. Thus, when restricting oneself to a factored policy, one may lose the ability to generate an optimal policy because one's decision ignores some part of the state space. This is why weaker notions of optimality, such as recursive and hierarchical optimality were introduced in the field of hierarchical reinforcement learning (Dietterich 2000; Andre and Russell 2002). In this paper, we show that under a certain assumption that applies to the type of domains that motivate this work, there exist a factored policy that is globally optimal.

The *Reset* Assumption

The extra property we introduce is the *reset* assumption. The intuitive idea is simple: every time we move control to one of the non-root sub-processes, the value of its private variables changes back to their initial value. This assumption basically means that we cannot start working on a task, move to another, and then come back to the first task and find it in the state we left it. Note that in general, it is restrictive. However, we believe it applies to a large sub-class of problems, including our rover domain (see below). This assumption may be modelled by adding to the $Abort$ action the effect of resetting the private variables of the current process to their initial values. Note that the non-private variables of a sub-process are not required to change in any particular way, and that the *reset* assumption implies the local policy μ_i is applicable only when all the variables in $\widetilde{X}_i$ (for $i > 0$) have their initial values. Finally, we mention a special case of the *reset* assumption, which we call the *visit-once* assumption. It stipulates that each sub-task is attempted once only. This assumption makes for a reasonable heuristic in many domains.

The Rover Application Domain

To illustrate these ideas, consider the problem of exploratory rovers. In this problem an autonomous vehicle, the rover, must visit a number of locations. Some locations contain an item of interest, e.g., a rock on which the rover can perform an experiment. A reward is obtained when an experiment concludes successfully. There is no value to repeating a successful experiment. Experiments usually involve instrument placements, preparation of the rock (e.g., coring), and measurements. Abstractly, the actions used at each of the locations are the same, e.g., extending the arm, placing this or that instrument, stowing the arm, etc. However, in practice, the transition functions for these actions depend on the position, structure, and nature of the rock. Thus, we have different instances of each instrument placement action for each location. Actions may fail and their resource consumption is uncertain.

We model the overall problem as follows: the root process describes the global status of the problem: the rover's location, the set of targets currently tracked, the state of instruments, resource levels, and which experiments were successful. The possible actions involve tracking different targets, navigating to different locations, and warming up instruments. The sub-processes describe the state of the rover and the experiment at a particular rock, as well as the part of the global information that is relevant to this task, such as resource levels. The actions correspond to the local manipulation of the rover's instruments. This decomposition is illustrated for a simplified rover problem with two sub-tasks in Figure 1.

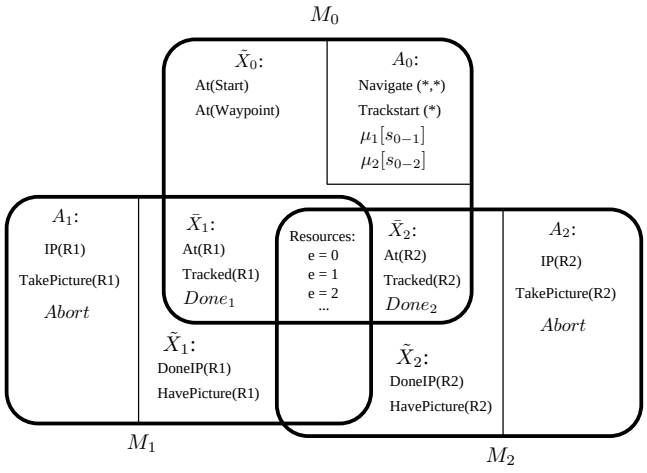


Figure 1: Hierarchical decomposition of a simplified rover problem: a root process M_0 navigates among two rocks, R1 and R2, and schedules two corresponding sub-processes M_1 and M_2 (IP stands for "Instrument Placement"). The hierarchy encapsulates the natural structure of the domain: the discretized resource is shared among all processes, and the part of the state specific to rock $i > 0$ is either private to M_i or shared by M_0 and M_i. The solving of M_i enriches the root process with a macro action μ_i that is an arm placement with conditions for aborting.

As mentioned, the *reset* assumption holds true for our current rover model, because the rover must have all of its instruments on board, and its arm stowed before it can move. Thus, we cannot leave an instrument in one location once we move to another location. Moreover, actions of preparing the rock for a measure, such as coring the rock, have to be re-done if we abort this rock before completing the measure, because it is not possible to put the rover arm exactly at the place it was when the rock was cored. So, all intermediate work towards the goal is lost once we move to another rock.

Algorithms

We now describe two algorithms for generating a factored policy in an SOSP. The fundamental, and well-known idea behind both algorithms is to repeatedly solve small parts of the problem that correspond to different sub-processes. What is new in our approach is the manner in which this is done, by exploiting the special structure of SOSPs to generate an optimal policy and to efficiently compute the macro-actions models. Our algorithms can be used with any MDP solution method to solve the different sub-processes.

Macro-actions play an essential role in this technique. Both algorithms augment the root domain with a macro operator $\mu_i[s_{0-i}]$, $i > 0$, $s_{0-i} \in S_{0-i}$, corresponding to execution of a local policy for the child domain M_i from states $s_0 = (s_{0-i}, \bar{s}_i) \in S_0$. This notation is used to emphasize the fact that the actual policy over S_i implemented by the macro may depend on the value of the local variables of M_0. The macro-operators terminate with the children's local *Abort* action that returns the control to the root process. Each macro plays the same role as an ordinary action of M_0.[2] When we apply macro $\mu_i[s_{0-i}]$, the value of s_{0-i} may not change. Thus, under the reset assumption, what characterizes a macro is the probability of ending up with some value of $\bar{S}_i$ given that we started with another. Therefore, macro-actions transition probability and rewards may be expressed as functions of the variables in $\bar{X}_i$ only. To distinguish these actions from the primitive actions, we denote their reward function by $\mathbf{R}_i(\bar{s}_i, \mu_i[s_{0-i}], \bar{s}'_i)$ and their transition function by $\mathbf{T}_i(\bar{s}_i, \mu_i[s_{0-i}], \bar{s}'_i)$.

In what follows, we assume for the sake of simplicity that the macros terminate after a finite time. This does not follow from our definition of SOSPs, nor is it essential – i.e., the algorithms and proofs below can be modified to handle non-terminating macros (and some work without modification). However, most domains we have in mind satisfy this property, either because the sub-MDPs are really stochastic shortest-path problems in which all actions have some positive probability of success (which implies the required property), or because actions consume resources with a positive probability (most typically with probability 1) and resources are eventually exhausted.

The Sub-Process Pairs Algorithm

In the sub-process pairs algorithm, we combine each child process with the root process, and solve them together. For each i, we define an MDP M_{0+i} with state space $S_{0+i} = 2^{X_0 \cup X_i}$ and action space

$$A_{0+i} = A_0 \cup A_i \cup \bigcup_{j \neq i; s_{0-j} \in S_{0-j}} \mu_j[s_{0-j}] \ .$$

That is, the actions available are the primitive actions of M_0 and M_i, plus all macro-actions for processes $j \neq i$.[3] Transition probabilities and rewards for M_{0+i} are directly derived from the definition of the SOSP, and from the transition probabilities and rewards of macro-actions μ_j, $j \neq i$. Bellman optimality equation for M_{0+i} may be written as:

$$V_{0+i}(s_{0+i}) = \max \left\{ \begin{array}{l} \max_{a_0 \in A_0} \left[\sum_{s'_0 \in S_0} R_0(s_0, a_0, s'_0) + T_0(s_0, a_0, s'_0) V_{0+i}(s'_0, \tilde{s}_i) \right] ; \\ \max_{a_i \in A_i} \left[\sum_{s'_i \in S_i} R_i(s_i, a_i, s'_i) + T_i(s_i, a_i, s'_i) V_{0+i}(s_{0-i}, s'_i) \right] ; \\ \max_{j \neq i} \left[\sum_{\bar{s}'_j \in \bar{S}_j} \mathbf{R}_j(\bar{s}_j, \mu_j[s_{0-j}], \bar{s}'_j) + \right. \\ \qquad \left. \mathbf{T}_j(\bar{s}_j, \mu_j[s_{0-j}], \bar{s}'_j) V_{0+i}(s_{0-j}, \bar{s}'_j, \tilde{s}^0_i) \right] \end{array} \right\}$$

[2] A macro is actually a temporally extended action (Sutton *et al.* 1999). If we were to use a discounted reward criteria, we would also need to model the expected duration of a macro. This would slightly complicate things, but the theory is well understood.

[3] In this algorithm, the *Abort* action is not needed.

where $s_{0+i} = (s_0, \tilde{s}_i) = (s_{0-i}, s_i) = (s_{0-j}, \bar{s}_j, \tilde{s}_i)$.

The algorithm goes through a loop solving MDPs M_{0+i} in an arbitrary order until steady-state. While solving M_{0+i}, a new policy $\mu_{0+i} : S_{0+i} \rightarrow A_{0+i}$ is determined. Consequently, the macro-actions for M_i are updated following

$$\mu_i[s_{0-i}](s_i) = \begin{cases} \mu_{0+i}(s_{0-i}, s_i) & \text{if } \mu_{0+i}(s_{0-i}, s_i) \in A_i \ , \\ \mathit{Abort} & \text{otherwise} \ . \end{cases}$$

Here is the pseudo-code of the algorithm:

1: Initialize each macro $\mu_i[s_{0-i}]$ to Abort everywhere.
2: **repeat**
3: **for** every sub-process pair M_{0+i} **do**
4: Solve M_{0+i};
5: Update macro-actions $\mu_i[s_{0-i}]$ based on the solution of step 4;
6: **until** no sub-policy has changed

Algorithm 1: Sub-Process Pairs Algorithm

Theorem 1. *Under the* reset *assumption, the sub-process pairs algorithm converges to a globally optimal policy in a finite number of iterations.*

Proof. Given a fixed strategy for breaking ties, the behavior of the algorithm does not depend on the technique used to solve each M_{0+i}. Therefore, the general result will be established if we prove the theorem in the particular case where Policy Iteration (Puterman 1994) is used at step 4 of the algorithm. The *reset* assumption implies that with each local state (s_{0-i}, s_i) of M_{0+i}, only a single global state is consistent (reachable). This is the state where all variables private to a process M_j, $j \neq i$ have their initial value:

$$G(s_{0-i}, s_i) = (s_{0-i}, \tilde{s}^0_1, \ldots, \tilde{s}^0_{i-1}, s_i, \tilde{s}^0_{i+1}, \tilde{s}^0_n) \in S \ .$$

The reason being that whenever we are in some sub-process, all other sub-processes must be in their initial local state. Thus, if we associate every local state of a sub-process with the corresponding reachable global state, we can view steps performed on a sub-process state (i.e., policy improvement and policy evaluation) as being performed in the global state. If we show that these steps converge in the global state space, we are done. Indeed, our algorithm is emulating a version of standard policy iteration on the complete state-space. To see this, recall that policy iteration works no matter how many states are updated in the policy improvement stage (as long as at least one of the possible improvements is performed) (Littman *et al.* 1995). The only condition is that each policy evaluation step produces an accurate value function. Moreover, after each stage of policy evaluation in M_{0+i}, the value function of that process-pair accurately represents the global value function over all states where sub-processes M_j, $j \neq i$ are in their initial condition:

$$V_{0+i}(s_{0-i}, s_i) = V(G(s_{0-i}, s_i)) \ .$$

This follows from standard results on planning with temporally abstract actions (Sutton *et al.* 1999). Note that the value function following the local policy evaluation phase of M_{0+i} is *not* accurate in those states where a sub-process M_j, $j \neq i$ is *not* in its initial states:

$$V_{0+j}(s_{0-j}, s_j) \neq V(G(s_{0-j}, s_j)) \ .$$

This is because the macro-actions $\mu_i[s_{0-i}]$ may have changed and V_{0+j} has not been update in the mean time. Fortunately, it is good enough, as for each sub-process pair M_{0+i}, we only update the actions in S_{0+i}, and we "jump over" all states not in S_{0+i} using macro-actions parameters $\mathbf{T}_j$ and $\mathbf{R}_j$, $j \neq i$. In summary, the algorithm may be seen as repeatedly performing: (i) policy evaluation over all global states of interest, and (ii) policy improvements only in global states where all sub-process M_j, $j \neq i$ are in their initial condition, until no further improvement is possible. Then the algorithm moves to the next process-pair. The algorithm terminates after a finite number of steps when no sub-policy has changed over one iteration. At this point, the value function of each sub-process pair accurately represents the global value function. □

The Abort-Update Algorithm

The sub-process pairs algorithms accounts for the (weak) coupling in between sub-tasks by always including the variables and actions of the root process when solving a sub-task. The abort-update algorithm solves each process M_i, $i \geq 0$ independently of the root process and ensures the synchronization in between sub-task by using different rewards for the Abort action. Intuitively, the Abort action should receive as a reward the value of the M_0 state where we will end-up after passing control back to the root. Therefore, we use the value function for M_0 to define the immediate reward for aborting M_i, $i > 0$. While this is the high-level story, the detailed picture is a bit more complicated. Consider the abort action for M_i. The value of aborting depends on what we can get from the other sub-processes. This is exactly the value of the current M_0 state. However, M_i can see only the variables in $\bar{X}_i$, and not those in X_{0-i}. Yet, the value of aborting depends on both.This means that we actually need to solve a version of M_i for each assignment to X_{0-i} because each such assignment would yield a potentially different value for *Abort*.[4]

The abort-update algorithm cyclically solves sub-processes M_i, $i \geq 0$ until a steady-state is reached. The Bellman equations of the root process is:

$$V_0(s_0) = \max \left\{ \max_{a_0 \in A_0} \left[\sum_{s'_0 \in S_0} R_0(s_0, a_0, s'_0) + T_0(s_0, a_0, s'_0) V_0(s'_0) \right] ; \right.$$
$$\left. \max_{i>0} \left[\sum_{\bar{s}'_i \in \bar{S}_i} \mathbf{R}_i(\bar{s}_i, \mu_i[s_{0-i}], \bar{s'}_i) + \mathbf{T}_i(\bar{s}_i, \mu_i[s_{0-i}], \bar{s'}_i) V_0(s_{0-i}, \tilde{s}'_i) \right] \right\}$$

[4] This is the prime motivation for using a different macro μ_i for each $s_{0-i} \in S_{0-i}$.

where $s_0 = (s_{0-i}, \bar{s}_i)$, and the Bellman equation of a leaf process is:

$$V_i[s_{0-i}](s_i) = \max \left\{ \max_{a_i \in A_i} \left[\sum_{s'_i \in S_i} R_i(s_i, a_i, s'_i) + T_i(s_i, a_i, s'_i) V_i[s_{0-i}](s'_i) \right] ; \; V_0(s_{0-i}, \bar{s}_i) \right\} .$$

In the second equation, the term $V_0(s_{0-i}, \bar{s}_i)$ represents the reward for aborting. Once the MDP $M_i[s_{0-i}]$ is solved, the macro-action $\mu_i[s_{0-i}]$ is updated to the optimal solution of this process. Here is the pseudo-code of the algorithm:

```
1: Initialize each macro μ_i[s_{0-i}] to Abort everywhere.
2: repeat
3:    Solve M_0
4:    for every sub-process M_i, i = 1,...,n do
5:       for every assignment s_{0-i} to X_{0-i} do
6:          Solve M_i[s_{0-i}];
7:          Update μ_i[s_{0-i}] based on the solution of step 6;
8: until no sub-policy has changed
```

Algorithm 2: Abort-Update Algorithm

Theorem 2. *Under the* reset *assumption, the abort-update algorithm converges to a globally optimal policy in a finite number of iterations.*

Proof. We can re-use most of the arguments developed in the proof of Theorem 1. First, it is sufficient to show that the theorem holds when Policy Iteration is used at step 3 and 6. Second, under the *reset* assumption, each state s_0 of M_0 represents the single reachable global state

$$G(s_0) = (s_0, \tilde{s}^0_1, \ldots, \tilde{s}^0_n) \in S ,$$

and each state of s_i of $M_i[s_{0-i}]$ represents the single reachable global state

$$G[s_{0-i}](s_i) = (s_{0-i}, \tilde{s}^0_1, \ldots, \tilde{s}^0_{i-1}, s_i, \tilde{s}^0_{i+1}, \tilde{s}^0_n) \in S .$$

Therefore, the algorithm may be seen as working on the set of reachable global states. Now, conversely to the sub-process pairs algorithm, abort-update may not be seen as an implementation of standard Policy Iteration in the global state space. In particular, the value function at the end of the policy evaluation stage of a process does not always represent exactly the value function of the global process. At the end of the policy evaluation stage of M_0, we have, as in the sub-process pairs algorithm,

$$V_0[s_0] = V(G(s_0)) .$$

This comes from the standard arguments on planning with temporally abstract action. Then we move to process M_1 and perform several iterations of policy evaluation followed by policy improvement. At the end of the first stage of policy evaluation, we still have the desired property:

$$V_1[s_{0-1}] = V(G[s_{0-1}](s_1)) .$$

This is due to the fact that the reward for aborting is equal to the value of the root process, which is an exact representation of the global value function. However the policy improvement stage modifies the macro-actions $\mu_1[s_{0-1}]$, and these changes are not propagated to the states of M_0. Therefore, at the second iteration of policy evaluation in M_1, we have

$$V_1[s_{0-1}] \neq V(G[s_{0-1}](s_1)) .$$

Therefore, the abort-update algorithm cannot be related to standard Policy Iteration in the global state space. Fortunately, we can establish a correspondence between the algorithm and another global algorithm that is known to converge. For the abort-update algorithm, we rely on the convergence of *Asynchronous Policy Iteration* (Bertsekas and Tsitsiklis 1997). In Asynchronous Policy Iteration, the policy evaluation and policy improvement steps are not synchronized. That is, the policy evaluation step is not necessarily carried until termination, and does not have to include all the states of the problem. Value updates on some states are interspersed with policy updates on some states. Bertsekas and Tsitsiklis (1997) show that this algorithm converges to an optimal policy. With the correspondence established between reachable global states and local states, it is now apparent that our abort-update algorithm is emulating a particular implementation of Asynchronous Policy Iteration. Indeed, (Bertsekas and Tsitsiklis 1997)[p.33] explicitly mentions the use of asynchronous policy iteration on a partitioned state space as a special case. □

The possible benefit of the abort-update algorithm over the sub-process pairs algorithm is that if the reward functions for two values of X_{0-i} are identical, we do not need to recompute the policy for M_i. Another advantage is that the cost per iteration can be substantially cheaper: The cost per iteration of dynamic programming on state space S is $O(|S|^2)$. Thus for the sub-process pairs algorithms, the complexity of an iteration is $O(|S_{0-i} \cup S_i|^2) = O(2^{2X_{0-i}+2X_i})$; for the abort-update algorithm, we solve M_i $|S_{0-i}|$ times, so the running time is $O(|S_{0-i}| \cdot |S_i|^2) = O(2^{X_{0-i}+2X_i})$. So overall, each iteration of Abort-Update is $O(|S_{0-i}|)$ times cheaper. As we will see later, our experimental results validate this expectation of improved performance.

Computing the Macro-Actions Parameters

One of the main steps in both algorithms is the computation of the macro transition probabilities $\mathbf{T}_i(\bar{s}_i, \mu_i, \bar{s}'_i)$ and expected reward $\mathbf{R}_i(\bar{s}_i, \mu_i, \bar{s}'_i)$ for a given macro-action $\mu_i : S_i \rightarrow A_i \cup \{Abort\}$.[5] This section discusses the computation of $\mathbf{T}_i$. Given that the reward is non-zero only if the macro reaches the goal before completion, $\mathbf{R}_i$ is easily deduced from $\mathbf{T}_i$.

Each macro defines a policy for its sub-process, which induces a Markov chain on S_i. There are well known methods for computing the state arrival probability in such cases (Kemeny and Snell 1976). Since we have assumed that macros

[5] In this section we omit the argument $[s_{0-i}]$ in macro-actions, since the computation is the same for all macro-actions.

complete in finite time with probability 1, this Markov chain is absorbing. The absorbing states are the states where $Abort$ is the optimal action (which includes the states where the goal is achieved). All the states that are non absorbing are called transient states. The transient states form one or several strongly connected components. If there are several such components, we say that the chain is *structured*. There is a natural ordering of the strongly connected components: every trajectory starts in a component and move irreversibly from component to component until it gets absorbed in an absorbing state. Assumptions on the planning domain determine the structure of the chain. For instance: (i) if resources are always decreasing or constant, then each resource level defines a strongly connected component; (ii) if, moreover, every local action may only advance the sub-task towards its goal or leave it unchanged (failure), then all loops of the chain are self-loops, and every transient state constitutes a different strongly connected component. The later case applies to the toy rover problem used in our simulations. This structure may be used to accelerate macro parameters computation.

We denote by T_{μ_i} the transition matrix of the Markov chain induced by μ_i. Given an initial value $\bar{s}_i \in \bar{S}_i$, there is a single consistent initial state for the chain, $(\bar{s}_i, \tilde{s}_i^0)$ (this follows from the *reset* assumption). The goal is then to compute, $\lim_{t\to\infty} T_{\mu_i}^t((\bar{s}_i, \tilde{s}_i^0), s_i')$ for all $s_i' \in S_i$ such that $\mu_i(s_i') = Abort$. Then we have

$$\mathbf{T}_i(\bar{s}_i, \bar{s}_i') = \sum_{\tilde{s}_i \in \tilde{S}_i} T_{\mu_i}^{\infty}((\bar{s}_i, \tilde{s}_i^0), (\bar{s}_i', \tilde{s}_i')) \, .$$

In this work, we consider two techniques for computing these values.

Forward technique: We denote by $\pi_i^t(s_i \mid \bar{s}_i)$ the probability of the chain being in state s_i after t transitions, knowing it started in state $(\bar{s}_i, \tilde{s}_i^0)$. It is easy to show that, for each absorbing state s_i^a, we have $T_{\mu_i}^{\infty}((\bar{s}_i, \tilde{s}_i^0), s_i^a) = \pi_i^{\infty}(s_i^a \mid \bar{s}_i)$ and

$$\pi_i^{\infty}(s_i^a \mid \bar{s}_i) = \sum_{s_i \in S_i} T_{\mu_i}(s_i, s_i^a) \sum_{t=0}^{\infty} \pi_i^t(s_i \mid \bar{s}_i) \, .$$

So, it is sufficient to compute $\sum_{t=0}^{\infty} \pi^t(s_i \mid \bar{s}_i)$ for each transient state s_i, and then push all this probability mass to the absorbing states in one step. Starting from the recursive equation:

$$\pi_i^{t+1}(s_i \mid \bar{s}_i) = \sum_{s_i' \in S_i} T_{\mu_i}(s_i, s_i') \pi_i^t(s_i' \mid \bar{s}_i) \ ,$$

we end up with the following system of linear equations with one equation and one unknown for each transient state:

$$\sum_{t=0}^{\infty} \Pi_i^t = T_{\mu_i} \sum_{t=0}^{\infty} \Pi_i^t + \Pi_i^0 \ ,$$

which has a unique solution

$$\sum_{t=0}^{\infty} \Pi_i^t = (I - T_{\mu_i})^{-1} \Pi_i^0 \ .$$

This system of linear equations can be solved analytically or by successive approximations. When the chain has additional structure, each strongly component may be treated independently as follows: We start by computing $\sum_{t=0}^{\infty} \pi_i^t(s_i \mid \bar{s}_i)$ for each s_i in the initial component. This is a smaller system of linear equations with one equation and one unknown for each state in the initial component. Next, we push all this probability mass to the successor components in one step, and solve each of them independently of the others, again, dealing with smaller systems of linear equations. Finally, when all strongly connected components are solved, we push all the probability mass to the absorbing states in one step. This is an application of standard technique to accelerate Gaussian elimination in structured systems of linear equation.

If all loops in the chain are self-loops, as in our toy problem, then all systems of equations above have a single unknown and a single equation which takes the form: $\sum_{t=0}^{\infty} \pi_i^t(s_i) = c_i + \Pr(s_i \mid s_i, \mu_i) \sum_{t=0}^{\infty} \pi_i^t(s_i)$, where $\Pr(s_i \mid s_i, \mu_i)$ is the probability of a self-loop in s_i under μ_i and c_i is some constant. So $\sum_{t=0}^{\infty} \pi_i^t(s_i) = c_i/(1 - \Pr(s_i \mid s_i, \mu_i))$. This allows for a very fast analytical computation of macro-action parameters in our toy problem.

Backward technique: The backward technique computes $T_{\mu_i}^{\infty}$ once and for all, and then extracts all the relevant information from it. We start from the recursive equation:

$$T_{\mu_i}^{\infty}(s_i, s_i^a) = \sum_{s_i' \in S_i} T_{\mu_i}(s_i, s_i') T_{\mu_i}^{\infty}(s_i', s_i^a) \ ,$$

where s_i^a is an absorbing state of S_i. Denoting $T_{\mu_i}[s_i^a]$ the vector of transition probabilities to the absorbing state s_i^a, we end up with the following system of linear equations:

$$T_{\mu_i}^{\infty}[s_i^a] = T_{\mu_i}[s_i^a] + T_{\mu_i} T_{\mu_i}^{\infty}[s_i^a] \ ,$$

which has the following unique solution

$$T_{\mu_i}^{\infty}[s_i^a] = (I - T_{\mu_i})^{-1} T_{\mu_i}[s_i^a] \ .$$

Again this system may be solved analytically or by linear approximations. If there is structure in the Markov chain, accelerated Gausian elimination techniques lead to the following backward algorithm: First, compute the transition probability $T_{\mu_i}^{\infty}(s_i, s_i^a)$ for all states s_i in the last strongly connected component(s) before absorbing states. This amounts to solving a linear system of dimension lesser than the above. Next, use this result to compute $T_{\mu_i}^{\infty}[s_i^a]$ in the second to last component(s), and so on. Again, if all loops are self loops, all systems have dimension one and we have $T_{\mu_i}^{\infty}(s_i, s_i^a) = c_i/(1 - \Pr(s_i \mid s_i, \mu_i))$.

Comparison: If the transition matrix T_{μ_i} is structured, then the forward technique may be implemented without an explicit representation of it (the algorithm manipulates only occupation probabilities $\sum_{t=0}^{\infty} \pi_i^t$). This can save considerable memory. On the other hand, the whole computation is repeated once for each initial value of $\bar{s}_i$. Therefore, if the chain can go through the same state s_i with two different initial values of $\bar{s}_i$, we implicitly compute the probability of the

paths from s_i to an absorbing state several times (once for each initial value of $\bar{s}_i$). However, this technique can identify early on that a state of the Markov chain is not reachable given the initial condition, and thus focus the computation only on the reachable states.

Conversely, the backward technique needs an explicit representation of the transition matrix $T^{\infty}_{\mu_i}$, which consumes memory. It never performs duplicate computation, since the probability of all paths leading to an absorbing state are computed once and for all. However, it is not able to identify that a state is not reachable given the initial conditions before the computation is complete, and so, it might compute the probability of some path from an unreachable state to an absorbing state.

Therefore, we have different trade-offs in terms of memory and execution time. In our simulations, we implemented the analytical version of both techniques. Our results show an advantage in terms of execution time to the forward technique in all the problem instances tried (there are five of them). This shows that in our domain model, reachability analysis saves us more time than what duplicate computations cost.

Empirical Results

We implemented and tested our two hierarchical algorithms using Value Iteration to solve sub-processes, as well as standard (flat) Value Iteration, on an instance of the rover domain. Although this is a simplified instance of the real domain used at NASA (e.g., continuous variables were discretized and there are no concurrent actions) (Bresina *et al.* 2002), it remains a challenging problem for current MDP solution algorithms. In this domain, sub-tasks representing rocks of scientific interest are completed by performing a number of successive actions. For instance, we must first deploy the rover arm, then position the rock abrasion tool, core the rock, position the camera, take the picture, and finally stow the arm. These actions consume uncertain amount of resources and may fail with positive probability. So, each sub-process represent a single chain of states with possible self-loops. At high level, we have to decide between navigating to a location, and, if there is a target at the current location, starting to work on that target. We varied the following parameters of the problem:

- the number of actions necessary to complete a sub-task, which directly determines the number of states in which each sub-process can be, but does not affect the size of the root process;
- the range of discretized resources. As resources represent shared variables, it determines the size of the separation between each sub-process and the root process;
- the total number of locations and targets, which determines the size of the root process but does not affect the size of each sub-process.

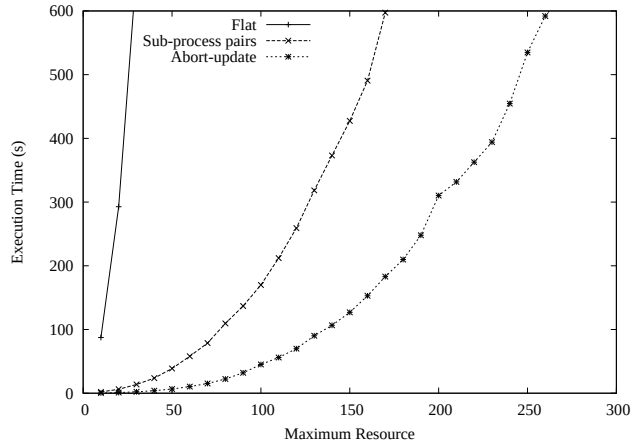
Simulation results are presented in Fig. 2 and 3. All the graphs in these figures represent execution time (in seconds) as a function of the resource range (the minimum resource is 0 in all cases). Figure 2(a) presents the overall computation time of each algorithm on the reference problem, which contains 5 sub-process with 6 states in each sub-process. In this experiment, both the abort-update and the sub-process pairs algorithm use the forward macro-action computation technique. Not surprisingly, Fig. 2(a) shows that the flat algorithm is largely outperformed by the hierarchical ones. Moreover, in this problem as in the 4 other problems we tested, abort-update exhibit better performances than the sub-process pairs. Figure 2(b) was obtained with the same problem instance, using the abort-update algorithm. It compares the performance of the two techniques for computing macro-actions parameters. It shows that the forward technique is slightly faster than the backward one. Again, qualitatively identical results were obtained with all problem instances tried. Figure 3 illustrate how the abort-update with forward macro-action computation is impacted by the size of the problem. As we see from Fig. 3(a), the algorithm is strongly influenced by the number of targets. This is where exponential growth is to be expected, and although this is much better than using a flat model, more work is needed to help the algorithm scale up to large numbers of sub-tasks. Finally, in Fig. 3(b) we see that the complexity of the local sub-tasks has much less influence on the computational effort, which is quite encouraging. These results also show that the solution of the root process is the dominant factor in the complexity of the algorithm.

Summary and Related Work

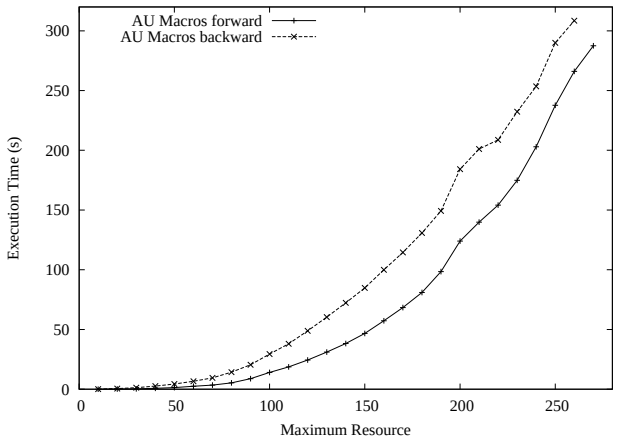
We describe an approach for solving Stochastic Oversubscription Planning problems. This approach exploits the hierarchical structure of the problem and works by iteratively solving naturally defined sub-problems. Under the *reset* assumption, these algorithm converge to a globally optimal policy.

Our method approach to SOSPs utilizes ideas that appeared in some of the previous work on problem decomposition studied in the area of decision-theoretic planning and the hierarchical reinforcement learning literature. Our solution approach starts with a hierarchical problem representation. (Amir and Engelhardt 2003) shows how such a decomposition can be constructed automatically from a given problem description for classical planning problems. We have implicitly used and extended these ideas to stochastic planning problems. Our domain imposes a natural two-level hierarchy, but deeper tree decompositions are possible.

Given a decomposition of the domain, it would be nice if the complexity of the solution algorithms would depend on the size of the local sub-domains, rather than the global state space. Unfortunately, as people have discovered in the past, this is *not* true in general. This leaves two avenues of research: (1) finding factored policies that satisfy weaker forms of optimality; and (2) finding additional restrictions that suffice for making factored policies optimal. Much work, especially in hierarchical reinforcement learning has pursued the first option. Starting with the MaxQ algorithm (Dietterich 2000), researchers have considered how to find optimal policies of particular form – one that conforms to some hierarchy of sub-routines, or more generally, to programs. The MaxQ algorithm yields a recursively optimal policy. This is a very weak local optimality guaran-



(a) Algorithms comparison.



(b) Macro parameters computation comparison.

Figure 2: Simulation results: algorithms comparison

tee, whose relation to global optimality is difficult to assess. Later (Andre and Russell 2002) showed how to obtain the more powerful level of hierarchical optimality. Hierarchical optimality is optimality with respect to the restricted class of policies that factors according to a pre-defined hierarchy. Thus, the work in hierarchical RL starts with hierarchical policy structure and attempts to find the best policies of this form. The restriction on the policy paves the way for the use of abstraction (because some aspects of the domains within the special class of policies used are irrelevant). For hierarchical optimality to hold, these "certificates" for these abstractions must be provided explicitly by the user.

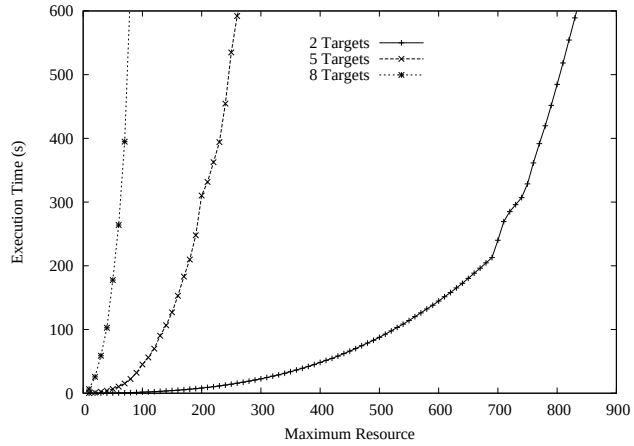
An important difference between that line of research and our work stems from the fact that we assume a domain model. With this model, a model-based decomposition is used, and it induces both natural abstractions as well as a natural factored form for the policy. We showed that under certain assumptions, optimality can be maintained under this factorization. It is interesting to note that (Andre 2003) discussed similar assumptions in the context of proving the convergence of a variant of Q-learning. Also, the hierarchical optimality of our solution follows immediately from these results with respect to the class of factored policies we seek even when the *reset* assumption is not satisfied.

As pointed out by (Guestrin and Gordon 2002), there are two common ways of splitting a problem into simpler pieces. *Serial decompositions* partition the state space into sub-regions. The canonical examples are robot activities involving navigation, where each sub-problem corresponds to some region of space. Much of the work done to date pursues this approach. A primary example, and one of the first ones is (Dean and Lin 1995), and at a high-level, our algorithm is very similar to their algorithm, i.e., we too use an iterative approach that propagates information between sub-domains and converges to an optimal solution. Another example is (Hauskrecht *et al.* 1998). The other, less often discussed, type of decomposition is *Parallel decompositions*, where the state space is the product of the sub-problems, rather than their union. Some examples of such decompositions include (Guestrin and Gordon 2002; Meuleau *et al.* 1998). Thus, the size of each problem is exponentially smaller than that of the original problem, offering much more potential savings than serial decompositions. As explained above, parallel decomposition techniques are unlikely to yield optimal solution because decisions are based on partial information. What we show is that under the *reset* assumption, the parallel and the serial decomposition are almost identical. Although syntactically our decomposition is parallel, because of reachability considerations, it is equivalent to a serial decomposition because few states are really reachable.

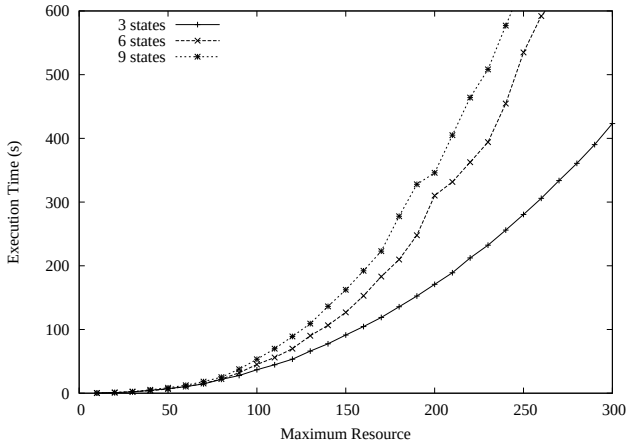
Overall, this paper offers a unified and principled model-based approach for using hierarchy and abstraction to tackle stochastic OSPs. Although the worst-case complexity of our algorithms remains exponential in the total number of domain variables, our initial experimental results show significant speed-ups in practice, indicating that this a promising approach that is likely to scale up to realistic domain models. It also shows that macro computation, long considered a very expensive steps, can be efficiently implemented given sufficient domain structure. We also see great potential for very fast approximately optimal algorithms (e.g., by ignoring small distinctions between macros for different states) and methods that make better use of reachability information.

References

E. Amir and B. Engelhardt. Factored planning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 929–935, 2003.



(a) Influence of the number of targets.



(b) Influence of the number of sub-process states.

Figure 3: Simulation results: influence of the problem size

D. Andre and S. Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 119–225, 2002.

D. Andre. *Programmable Reinforcement Learning*. PhD thesis, University of California, Berkeley, 2003.

D.P. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1997.

R. I. Brafman and Y. Chernyavsky. Planning with goal preferences and constraints. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, 2005.

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

T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1993.

T. Dean and S.H. Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1121–1129, 1995.

T. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of AI Research*, 13:227–303, 2000.

C. Guestrin and G. Gordon. Distributed planning in hierarchical factored MDPs. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 197–506, 2002.

S. Hanks and D. V. McDermott. Modeling a dynamic and uncerain world i: Symbolic probabilistic reasoning about change. *Artificial Intelligence*, 66(1):1–55, 1994.

M. Hauskrecht, N. Meuleau, L.P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 220–229, 1998.

J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer-Verlag, New York, NY, 1976.

M.L. Littman, T.L. Dean, and L.P. Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 394–402, 1995.

N. Meuleau, M. Hauskrecht, K.E. Kim, L. Peshkin, L.P. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 165–172, 1998.

M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, NY, 1994.

R. Sanchez and S. Kambhampati. Planning graph heuristics for selecting objectives in over-subscription planning problems. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, 2005.

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

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