Automated Resource-Driven Mission Phasing Techniques for Constrained Agents

Jianhui Wu
EECS Department, University of Michigan
Ann Arbor, MI 48109 USA
jianhuiw@umich.edu

Edmund H. Durfee
EECS Department, University of Michigan
Ann Arbor, MI 48109 USA
durfee@umich.edu

ABSTRACT

A constrained agent is limited in the actions that it can take at any given time, and a challenging problem is to design policies for such agents to do the best they can despite their limitations. One way of improving agent performance is to break larger tasks into phases, where the constrained agent is better able to handle each phase and can reconfigure its limited capabilities differently for each phase. In this paper, we present algorithms for automating the process of finding and using mission phases for constrained agents. We analyze several variations of this problem that correspond to different classes of important constrained-agent problems, and show through analysis and experiments that our techniques can increase an agent's rewards for varying levels of constraints on the agent and on the phases.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE; I.2.8 [**ARTIFICIAL INTELLIGENCE**]: Control Methods, and Search

General Terms

ALGORITHMS

Keywords

Mission phasing, constrained MDPs, abstract MDPs, mixed integer programming

1. MOTIVATION AND INTRODUCTION

Markov decision processes (MDPs) provide a good framework to compute optimal policies for autonomous agents operating in uncertain environments. However, the optimal policies computed by MDPs might not be executable by constrained agents. For example, a real-time autonomous driving agent might be unable to schedule all of its desired actions (watching for pedestrians, checking surrounding traffic, reading gauges, etc.) frequently enough to optimally reach its destination at maximum speed and minimum risk because it cannot redirect its limited perceptual resources fast enough in all relevant directions. Or, as another example, a Mars Rover might be unable to carry all of the tools desired by an optimal scientific mission on a given day.

Coping with agent architectures that constrain executable policies has been the subject of several recent studies. Altman has adopted a Lagrangian and dual LP approach to solve constrained MDPs with total cost criteria [1]. Feinberg has analyzed the complexity of constrained discounted MDPs [7]. Dolgov and Durfee presented algorithms for a wide range of constrained optimization problems via reduction to linear and integer programming [5, 6]. These approaches search for a policy that is executable within the agent constraints and that optimizes the expected (possibly discounted) reward accrued over the entire agent execution.

In constrained MDPs, it is generally the case that an executable policy is optimal with respect to the (probability distribution over the) state(s) in which the agent begins executing the policy [5, 6]. Not surprisingly, this suggests that a constrained agent might be able to do better if it can change its policy during execution: when it reaches a particular state, it might adopt a new policy that uses its constrained resources more effectively given the particular trajectory the world has taken. For example, when our autonomous driving agent reaches a highway entrance ramp, it might improve its expected reward by adopting a policy that calls for greater speed and more frequent traffic checks, balanced by less frequent checks for pedestrians and traffic lights. Our Mars Rover might discover that its path has brought it near to the lander, such that it might drop off instruments that it no longer needs and pick up others that might increase the value of its remaining science experiments for the day.

Thus, unlike an unconstrained agent that can execute a policy that is optimal for all possible eventualities, we argue that a constrained agent can benefit from judiciously breaking its overall mission into phases, where as it moves from phase to phase it can adopt a different, more effective policy for its current phase. This assertion is not surprising. The challenge, though, is in automating the process of creating and exploiting mission phases, as opposed to having the phases predefined in the description of the mission, in complex stochastic domains.

In contrast to techniques that improve computational tractability by decomposing larger planning problems into sequences of smaller problems (for example, passing through

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

intermediate *landmark* states [9, 15]), our use of mission phasing addresses execution constraints rather than computational constraints. Indeed, as will be seen, in general our mission phasing methods will not in themselves reduce computational requirements because a policy in one phase can only be optimized with respect to the policies planned for the possible subsequent phases that might be entered.

In this paper, we define the mission phasing problem (MPP) in stochastic domains as the problem of determining the states at which phases change (where an agent switches its policy) and the optimal policies to adopt for each phase (at each policy-switching state), given the agent constraints. We look at several increasingly general variations of the MPP, and for each, we present, analyze, and illustrate solution algorithms. We empirically evaluate the effectiveness of our techniques under varying degrees of agent constraints. Our presentation begins, though, by summarizing the concepts behind (constrained) Markov Decision Processes that our work builds on, and ends with a discussion of challenges that remain.

2. MDP AND CONSTRAINED MDP

A classical MDP can be defined as a four-tuple $\langle S, A, P, R\rangle$, where S is a finite state space, A is a finite action space, $P = \{p_{ij}^a\}$ is the state transition probability where p_{ij}^a is the probability that the agent reaches state j if it executes action a in state i, and $R = \{r_{ia}\}$ is the reward function where r_{ia} is the reward that the agent receives if it executes action a in state i.

In this paper, we concentrate on decision problems in transient Markov processes [10], although in general our results will apply to any contracting MDP. In a transient Markov process, an agent will eventually leave the corresponding Markov chain, after running a policy for a finite number of steps. In other words, given a finite state space, it is assumed that the agent visits any state only a finite number of times for any policy.

Well-known value and policy iteration algorithms are commonly used in solving unconstrained MDPs [16]. However, it is surprisingly hard to make these algorithms work on constrained problems, and many researchers adopt an alternative formulation that is based upon linear programming [1, 2, 5]. In this section, we recap how to formulate an unconstrained MDP into a linear program, whose solution yields the optimal policy maximizing the total expected reward, because our subsequent techniques extend these foundations.

If x_{ia} denotes the expected number of times action a is executed in state i, then the objective function $\sum_i \sum_a x_{ia} r_{ia}$ represents the total expected reward. Now, consider the linear program shown in Eq.1.

$$\max \sum_i \sum_a x_{ia} r_{ia} \tag{1}$$

subject to the following constraints:

$$\left| \begin{array}{l} \sum_a x_{ja} = \alpha_j + \sum_i \sum_a p_{ij}^a x_{ia} \\ x_{ia} \geq 0 \end{array} \right.$$

where α_j is probability that the system is initially in state j. The constraint $\sum_a x_{ja} = \alpha_j + \sum_i \sum_a p_{ij}^a x_{ia}$ indicates that the expected number of times state j is visited must equal the initial probability distribution at state j plus the expected number of times state j is entered via all possible transitions.

If we solve the linear program in Eq.1, it is trivial to derive the optimal policy that specifies the action(s) to take in a given state. Specifically, assigning a probability of executing action a at state i as $\pi_{ia} = \frac{x_{ia}}{\sum_a x_{ia}}$ will maximize the total expected reward. If any π_{ia} has a value other than zero or one, the optimal policy is randomized; otherwise it is deterministic.

Formulating unconstrained MDPs as linear programs makes it easier to add other constraints. A wide range of such constrained optimization problems have been investigated by Dolgov and Durfee [5, 6]. In order to familiarize readers with some background knowledge on solving constrained Markov decision problems, we show a simple example below. Let us say that the cost of incorporating action a into the policy is u_a (e.g., watching for pedestrians uses the camera 20% of the time), and the total cost cannot exceed an upper bound ω (e.g., the camera cannot be used more than 100% of the time). The linear program formulation of MDPs makes it easy to incorporate these kinds of costs and limitations by imposing the following constraint on x_{ia}

$$\sum_a u_a \theta(\sum_i x_{ia}) \leq \omega \tag{2}$$

where $\theta(z)$ is a step function, defined as

$$\theta(z) = \left\{ \begin{array}{ll} 1 & z > 0 \\ 0 & otherwise \end{array} \right.$$

Note that $\theta(z)$ is a nonlinear function, and so we need to convert the nonlinear constraint (Eq.2) into linear constraints for the sake of using linear or integer programming algorithms. One way to accomplish this is to introduce additional integer variables [5]. The details are shown below.

$$\left| \begin{array}{l} \sum_a u_a \Delta_a \leq \omega \\ \frac{\sum_i x_{ia}}{X} \leq \Delta_a \\ \Delta_a \in \{0, 1\} \end{array} \right. \tag{3}$$

where X is a constant greater than $\sup \sum_i x_{ia}$, and Δ_a, an integer in the interval [0, 1], is used to indicate whether action a is scheduled in the policy.

As a result, the constrained MDP is formulated as a mixed integer program. Mixed integer programming (MIP) is the discrete version of linear programming with an additional requirement that partial variables must be integers. MIPs can be solved by a variety of highly optimized algorithms and tools [4, 19]. In recent years, there has been substantial progress on using MIPs in automated planning [3, 11, 17, 18].

We illustrate this technique using the example shown in Figure 1; we will illustrate our subsequent extensions to these ideas using this example as well to help the reader understand and compare the techniques. In this example, there are six states $\{S_1, S_2, S_3, S_4, S_5, S_6\}$ and six actions $\{a_1, a_2, a_3, a_4, a_5, a_6\}$, where the agent starts at S_1, and a_1 is a *noop* that represents the fact that the agent has the freedom of not executing any action ($u_{noop} = 0$). The cost (u_a) of each action in the set $\{a_2, a_3, a_4, a_5, a_6\}$ is 1. If the cost upper bound ω is unlimited, this is an unconstrained MDP. Using policy iteration or linear programming (Eq.1), we could easily compute the optimal policy, which is $[S_1 \rightarrow a_2, S_2 \rightarrow noop/a_3, S_3 \rightarrow a_4, S_4 \rightarrow a_5, S_5 \rightarrow a_6, S_6 \rightarrow noop]$, and the total expected reward is 174.65.

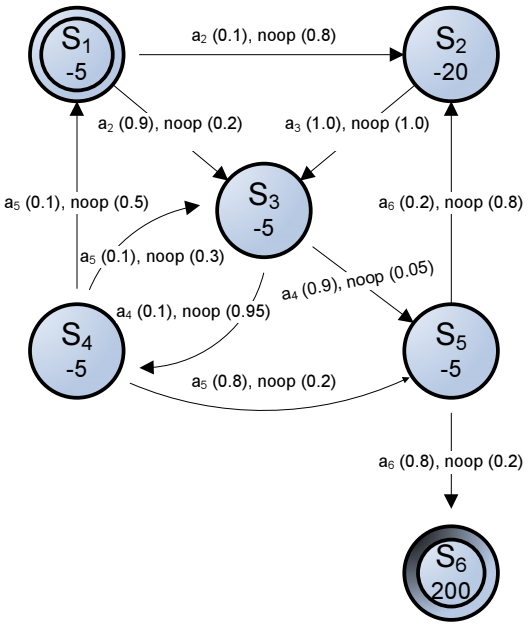

Figure 1: A simple example.

Suppose instead that the agent is highly constrained, such that now the agent can only execute a policy with one action that is not a *noop* ($\omega = 1$). It is not obvious which action the agent should schedule. We can solve the problem by formulating it as the following MIP (combining Eq.1 and Eq.3):

$$\max \sum_i \sum_a x_{ia} r_{ia} \tag{4}$$

subject to the following constraints:

$$\left| \begin{array}{l} \sum_a x_{ja} = \alpha_j + \sum_i \sum_a p_{ij}^a x_{ia} \\ x_{ia} \geq 0 \\ \sum_a u_a \Delta_a \leq \omega \\ \frac{\sum_i x_{ia}}{X} \leq \Delta_a \\ \Delta_a \in \{0, 1\} \end{array} \right.$$

where p_{ij}^a and r_{ia} are as shown in Figure 1, $\alpha_1 = 1$, $\alpha_{j \in \{2,\ldots,6\}} = 0$, $u_1 = 0$, $u_{a \in \{2,\ldots,6\}} = 1$, $\omega = 1$, and X can be computed by solving:

$$X = \max \sum_i \sum_a x_{ia} \left| \begin{array}{l} \sum_a x_{ja} = \alpha_j + \sum_i \sum_a p_{ij}^a x_{ia} \\ x_{ia} \geq 0 \end{array} \right. \tag{5}$$

Eq.5 yields X=70.24, which is guaranteed to be no less than $\sup \sum_i x_{ia}$, and Eq.4 is solved as:

$$[(x_{11}, x_{12}), (x_{21}, x_{23}), (x_{31}, x_{34}), (x_{41}, x_{45}), (x_{51}, x_{56}), x_{61}]$$
$$= [(3.47, 0), (3.03, 0), (5.21, 0), (4.95, 0), (0, 1.25), 1]$$
$$[\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6] = [1, 0, 0, 0, 0, 1]$$

The optimal policy is $[S_1 \rightarrow noop, S_2 \rightarrow noop, S_3 \rightarrow noop, S_4 \rightarrow noop, S_5 \rightarrow a_6, S_6 \rightarrow noop]$, and the corresponding total expected reward is reduced to 65.02 due to the constraints on total action costs. This is the optimal policy for the constrained agent that uses a single policy throughout its entire mission. We will use this example as we go along to see the degree to which automated mission phasing can improve this expected reward.

3. FIXED POLICY-SWITCHING STATES

We begin our examination of automated mission phasing by first assuming that policy-switching states are known *a priori*. This assumption fits many problems where the opportunity to switch policies is dictated by the state of the world rather than being a choice of the agent. In the case of a driving agent, for example, red lights and highway ramps are good opportunities for switching policies compared to trying to switch policies in fast-paced situations such as while careening among a crowd of moving pedestrians.[1] In the case of a Mars Rover, the locations on Mars where it can change its instrumentation will likely be very limited, and well known to it.

Decomposition techniques for planning in stochastic domains are widely used for large environments with many states [8, 14]. In those approaches, states are partitioned into smaller regions, a policy is computed for each region, and then these local policies are pieced together to obtain a global policy. Automated mission phasing techniques are analogous to decomposition techniques – partitioning a mission into multiple phases leads to smaller state/action spaces in each phase – though the motivation for mission phasing is the constraints on policies agents can execute rather than the reduction of computational requirements during policy formulation. Nonetheless, we can exploit these ideas. In this section, we assume that none of constraints is associated with more than one phase and postpone the discussion of more general constraints to Sections 4 and 5.

We now present an abstract MDP algorithm for solving MPPs. An abstract MDP is composed of abstract states, each of which represents a mission phase. The "action" for an abstract state is the policy used in its corresponding mission phase. Since it is assumed that agent constraints in one phase are unaffected by policy choices in another phase, the abstract MDP is an unconstrained MDP even though internally each phase is still a constrained MDP. The algorithm thus uses a policy iteration approach at the abstract level with an embedded MIP solver. The embedded MIP solver finds possible executable policies and their expected rewards for each of the phases, but different policies will have different probabilities of reaching the various policy switching states at the "edges" of the phase. The outer policy iteration algorithm at the abstract level iteratively searches for the combination of phase policies that maximizes reward across the whole mission.

The detailed procedure for the abstract MDP solver is illustrated below:

1. *Partitioning the mission into smaller phases*
   When policy-switching states are given, partitioning a mission into multiple phases is trivial. Start from a state with a positive initial probability distribution, which we call START state, or a policy-switching state, and then keep expanding through all connected transitions until encountering other policy-switching states.

[1]Some real-time agent architectures, like CIRCA (which will be discussed later), explicitly model time-critical transitions, making it possible to identify states where a pause while switching policies could be catastrophic.

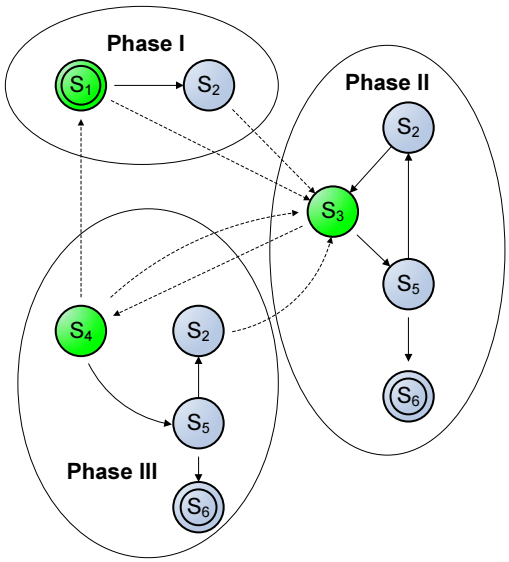

Figure 2: Abstract MDP with three phases.

2. *Policy Iteration*
   The following policy iteration algorithm is used, after the mission is partitioned.

   (a) Solve the corresponding unconstrained MDP and compute a utility function $U(s)$ for each policy-switching state s. $U(s)$ are used as initial values of policy-switching states in the MPP since they are likely to provide good estimates.

   (b) In the abstract MDP, each phase is treated as an abstract state and each policy for a phase is treated as an abstract action for that phase's abstract state. The policy iteration algorithm alternates between the following two steps:

   Policy evaluation: Given abstract actions, calculate $U(s)$ for each policy-switching state s. For small state spaces, standard linear algebra methods are often the best solutions for policy evaluation. For large state spaces, a simplified value iteration algorithm might be preferred (simplified because the policy in each phase is fixed).

   Policy improvement: Rather than enumerating all possible policies (abstract actions) for a phase (abstract state), the algorithm uses a constrained MDP solver, such as in Eq.4, to calculate the optimal policy in the phase, given the current utility functions of the (outgoing) neighboring policy-switching states.

We now return to our running example introduced in Section 2 to illustrate how the total expected reward can be improved if the agent can switch its policy at some states. Let us say that the agent knows it is able to switch policies at states S_1, S_3 and S_4. The corresponding abstract MDP is constructed and shown in Figure 2, which is composed of three abstract states. Using the above policy-iteration algorithm, and assuming the same parameters (especially that an executable policy cannot have more than one action that is not a *noop*), the utility functions of the policy-switching states eventually converge to $U(S_1) = 113.65$, $U(S_3) = 120.65$, and $U(S_4) = 123.05$. The optimal policy in phase I is $[S_1 \rightarrow a_2, S_2 \rightarrow noop]$, the optimal policy in phase II is $[S_2 \rightarrow noop, S_3 \rightarrow noop, S_5 \rightarrow a_6, S_6 \rightarrow noop]$ and the optimal policy in phase III is $[S_2 \rightarrow noop, S_4 \rightarrow noop, S_5 \rightarrow a_6, S_6 \rightarrow noop]$; the total expected reward is 113.65.

Thanks to the policy iteration algorithm, the abstract MDP solver generally converges quickly. However, it should be noted that two limitations are inherent in the abstract MDP solver. One of the limitations is that the abstract MDP solver requires that policy-switching states are known *a priori*. An approach to alleviate this limitation is to integrate the abstract MDP solver with search techniques, such as branch and bound algorithms. The other limitation is due to the possible existence of constraints running across multiple phases. The abstract MDP is an unconstrained MDP and so the embedded policy iteration algorithm is efficient and well-suited. In other words, the abstract MDP solver cannot cope with constraints associated with multiple abstract states, such as limitations on the total expected costs across phases, and restrictions on the expected number of visits to a specific state. In the following sections, we present a more general solver.

4. FIXED NUMBER OF POLICY SWITCHING STATES

In this section, we assume that policy-switching states are not given *a priori* and so an agent has to determine for itself an optimal set of policy-switching states. Obviously, if an agent could switch policies at every state, then it could obtain the same reward as in the unconstrained case (assuming that there is no action whose cost all by itself exceeds the policy cost constraint). In practice, though, a constrained agent will have limitations in how many contingent policies it can compute and/or store. Specifically, in this section we will assume an agent has an upper bound on the number of mappings it can represent between policy-switching states and the policy to adopt for each. The agent thus must decide which states and policies to put into this mapping.

When the number of policy-switching states must be less than a constant integer value K, we refer to the mission phasing problem as a constrained optimization MPP. Clearly, the abstract MDP solver presented in Section 3 cannot be used for the constrained optimization MPP because now a decision to create a policy-switching state in one phase will consume an entry in the state-to-policy mapping that will now be unavailable for another phase. That is, there are now constraints that span multiple phases.

In this section, we construct a mixed integer program, the solution to which yields the optimal set of policy-switching states maximizing the total expected reward. We make a simplifying assumption that a START state (which has a positive initial probability distribution α_j) is always a policy-switching state. This assumption makes the presentation clearer and representation more concise, as well as sidestepping the question of what the "default" agent policy might be (since that is what it would use if it could not switch policies in a START state).

Let x_{ia}^k be the expected number of times action a is ex-

ecuted in state i within phase k. Clearly, if state i is not reachable in phase k, then $x_{ia}^k = 0$. Let $\alpha_j^k = \sum_a x_{ja}^k - \sum_i \sum_a p_{ij}^a x_{ia}^k$, then α_j^k provides a way to characterize transitions among phases. If state j is neither a START state nor a policy-switching state, then $\alpha_j^k = 0$ for any k, since in any phase the expected number of times of visiting state j ($\sum_a x_{ja}^k$) must equal the expected number of times of entering state j through all possible transitions ($\sum_i \sum_a p_{ij}^a x_{ia}^k$). If state j is a policy-switching state, $\sum_k \alpha_j^k = \alpha_j$. Recall that α_j is the initial probability distribution for state j. $\sum_k \alpha_j^k = \alpha_j$ guarantees that the total expected number of times of visiting state j must equal the initial probability distribution for state j plus the total expected number of times of entering state j through all possible transitions.

Now, we can formulate the constrained optimization MPP into a mixed integer program shown in Eq.6. The objective function $\sum_i \sum_a \sum_k x_{ia}^k r_{ia}$ is the total expected reward, and K is the maximum number of policy-switching states.

$$\max \sum_i \sum_a \sum_k x_{ia}^k r_{ia} \tag{6}$$

subject to the following constraints:

$$\left|\begin{array}{ll} \sum_a x_{ja}^k - \sum_i \sum_a p_{ij}^a x_{ia}^k = \alpha_j^k & (cons.1) \\ \sum_k \alpha_j^k = \alpha_j & (cons.2) \\ X \geq \sup \alpha_j^k & (cons.3) \\ \frac{\alpha_j^k}{X} \leq \Psi_j & (cons.4) \\ \sum_j \Psi_j \leq K & (cons.5) \\ x_{ia}^k \geq 0 & (cons.6) \\ \Psi_j \in \{0,1\} & (cons.7) \\ k \in \{1,2,\ldots,K\} & (cons.8) \\ \text{other constraints} & (cons.9) \end{array}\right.$$

- As illustrated, constraint (1) in Eq.6 models the conservation of probability within a phase.

- Constraint (2) simply indicates the fact that $\sum_k \alpha_j^k = \sum_k (\sum_a x_{ja}^k - \sum_i \sum_a p_{ij}^a x_{ia}^k) = \sum_a x_{ja} - \sum_i \sum_a p_{ij}^a x_{ia} = \alpha_j$, where $x_{ia} = \sum_k x_{ia}^k$ is the total expected number of times action a is executed in state i.

- Constraint (3) defines $X \geq \sup \alpha_j^k$, which is used to guarantee $\frac{\alpha_j^k}{X} \leq 1$. Directly computing $\sup \alpha_j^k$ might not be easy. In transient systems, a feasible alternative is to compute $X = \max \sum_i \sum_a x_{ia}$, since

$$\begin{aligned} \sup \alpha_j^k &= \sup(\sum_a x_{ja}^k - \sum_i \sum_a p_{ij}^a x_{ia}^k) \\ &\leq \sup \sum_a x_{ja}^k \\ &\leq \sum_i \sum_a \sum_k x_{ia}^k \\ &\leq X \end{aligned}$$

  where $X = \max \sum_i \sum_a x_{ia}$ can be computed by using Eq.5.

- Ψ_j in constraint (4) is a binary variable, where $\Psi_j = 1$ when state j is a policy-switching state, and $\Psi_j = 0$ otherwise. Clearly, constraints (4) and (7) mean that $\exists k \; \frac{\alpha_j^k}{X} > 0 \Rightarrow \Psi_j = 1$.

- Constraint (5) says that the number of policy-switching states must be no greater than K.

- Constraints (6-8) denote the ranges of variables. Note that there is no restriction for the range of α_j^k.

- The previous constraints capture constraints on the number of policy switching states, but not other constraints such as constraints on executable policies (such as the total costs of the actions in a policy). Constraint (9) in the formulation is a placeholder for these other constraints (an example of these follows soon).

We now show how to derive an optimal MPP policy from the solution to Eq.6. The computation of the MPP optimal policy involves two steps:

1. *Computing the optimal policy in each phase k.* This is the same as before – at state i, action a is executed with probability $\pi_{ia}^k = \frac{x_{ia}^k}{\sum_a x_{ia}^k}$. Let $\pi^k = \{\pi_{ia}^k\}$ denote the phase policy in phase k.

2. *Determining which phase policy to adopt at a policy-switching state i.* This is also trivial. The agent should choose the phase policy π^k with probability $\Pi_i^k = \frac{x_i^k}{\sum_k x_i^k}$ at state i for maximizing the total expected reward, where $x_i^k = \sum_a x_{ia}^k$.

   Recall that constraint (9) is a placeholder for other constraints, which can vary in a wide range. In many MPPs (including our examples shown in this paper), the choice of policy at a policy-switching state is deterministic. However, for some cases (e.g., there exists execution constraints [5] across multiple phases), the choice of policy might be randomized. For these cases, if the deterministic choice is desired, we need to force that each policy-switching state only belongs to one phase. That is, the following constraints should be incorporated into Eq.6.

   $\sum_k \theta(x_i^k) \leq \left\{ \begin{array}{ll} 1 & \text{if state } i \text{ is a policy-switching state} \\ K & \text{otherwise} \end{array} \right.$

   where $\theta(z)$ is a step function, defined as $\theta(z) = 1 \Leftrightarrow z > 0$, $\theta(z) = 0 \Leftrightarrow z \leq 0$. This nonlinear constraint can be reformulated into integer constraints

$$\left|\begin{array}{l} \frac{x_i^k}{X} \leq \sigma_i^k \\ \sum_k \sigma_i^k \leq (1-K)\Psi_i + K \\ \sigma_i^k \in \{0,1\} \end{array}\right.$$

   where σ_i^k is a binary variable. $\sigma_i^k = 1$ when state i belongs to phase k, and $\sigma_i^k = 0$ otherwise.

We conclude this section by illustrating our solution on our running example. Recall that, as shown in Section 3, when the agent is allowed to switch its policy at S_1, S_3 and S_4, its total expected reward is 113.65. Rather than predetermining the policy-switching states, we now say that two additional policy-switching states besides START state S_1 can be chosen by the agent from any states in the system. The problem can then be solved by placing the following constraints (which represent a multi-phase version of Eq.3) into "other constraints" in Eq.6:

$$\left|\begin{array}{l} \sum_a u_a \Delta_a^k \leq \omega \\ \frac{\sum_i x_{ia}^k}{X} \leq \Delta_a^k \\ \Delta_a^k \in \{0,1\} \end{array}\right. \tag{7}$$

Using the same transition probability p_{ij}^a, reward r_{ia}, initial probability distribution α_j, utility cost u_a, the cost upper bound ω and constant value X as in Section 2, the integer solution of the mixed integer program is:

$$[\Psi_1, \Psi_2, \Psi_3, \Psi_4, \Psi_5, \Psi_6] = [1, 0, 1, 0, 1, 0]$$

$$\begin{vmatrix} \Delta_1^1, \Delta_2^1, \Delta_3^1, \Delta_4^1, \Delta_5^1, \Delta_6^1 \\ \Delta_1^2, \Delta_2^2, \Delta_3^2, \Delta_4^2, \Delta_5^2, \Delta_6^2 \\ \Delta_1^3, \Delta_2^3, \Delta_3^3, \Delta_4^3, \Delta_5^3, \Delta_6^3 \end{vmatrix} = \begin{vmatrix} 0, 1, 0, 0, 0, 0 \\ 0, 0, 0, 1, 0, 0 \\ 0, 0, 0, 0, 0, 1 \end{vmatrix}$$

The total expected reward of the agent is 173.80 by adopting the policy $[S_1 \rightarrow a_2, S_2 \rightarrow noop]$ at S_1, switching to the policy $[S_3 \rightarrow a_4, S_4 \rightarrow noop]$ at S_3, and switching to the policy $[S_2 \rightarrow noop, S_5 \rightarrow a_6, S_6 \rightarrow noop]$ at S_5.

5. INCORPORATING COST FOR POLICY SWITCHING

Now we neither assume that the policy-switching states are predefined (Section 3) or that the number of policy-switching states is bounded (Section 4). Instead, we assume that policy switching can occur at any state, and at as many states as desired, but that there is a cost associated with switching a policy at a state (and different states can have different costs). In our Mars Rover example, for instance, the Rover might be able to redistribute instruments around its environment so that it could switch policies when reaching any of those depots, but there would be a cost to creating those states.

Similarly, a number of problems in the real-time AI literature can fit into this case. For example, the Cooperative Intelligent Real-Time Control Architecture (CIRCA) [12, 13] provides a constrained agent architecture for time-critical applications. A CIRCA agent schedules a periodic set of tasks, where the tasks can gather sensor data, assess whether the system state requires an associated response, and if so take the response. The more states and responses the agent needs to be prepared to take, the less frequently it can do them all. Clearly, breaking a mission into smaller, simpler phases is one way to enable real-time responsiveness in each phase. However, switching policies cannot happen instantaneously; while CIRCA is detecting that it should switch policies and is making the switch, it might not monitor quite as frequently for events that it should respond to. To prevent missing critical responses, a CIRCA agent could employ ideas analogous to the Rover example by changing some of the states in the world to buffer itself from rapid real-time responses. In the driving domain, for example, long highway entrance and exit ramps are instances of such modifications that provide relatively benign states for switching policies.

For these kinds of problems, we associate a cost with making a state into one that is conducive to policy switching. If these costs are calibrated with the costs and rewards associated with executing policies, then the optimization problem is to maximize the total expected reward, accounting for the costs of creating policy-switching states, without predetermining which states or how many there will be.

We call these kinds of MPPs optimization MPPs, which can also be formulated into MIPs (Eq.8) in a similar manner as described in Section 4.

$$\max \sum_i \sum_a \sum_k x_{ia}^k r_{ia} - \sum_i c_i \Psi_i \qquad (8)$$

subject to the following constraints:

$$\begin{vmatrix} \sum_a x_{ja}^k - \sum_i \sum_a p_{ij}^a x_{ia}^k = \alpha_j^k \\ \sum_k \alpha_j^k = \alpha_j \\ X \geq \sup \alpha_j^k \\ \frac{\alpha_j^k}{X} \leq \Psi_j \\ \sum_j \Psi_j \leq N \\ x_{ia}^k \geq 0 \\ \Psi_j \in \{0, 1\} \\ k \in \{1, 2, \ldots, N\} \\ \text{other constraints} \end{vmatrix}$$

where c_i is the cost of allowing the agent to switch policies at state i, the objective function $\sum_i \sum_a \sum_k x_{ia}^k r_{ia} - \sum_i c_i \Psi_i$ is the total expected reward of the agent minus the cost for making policy-switching states, and N is the total number of states.

We now revisit our running example to illustrate how the above algorithm can be used to solve optimization MPPs. Let us say that $[c_1, c_2, c_3, c_4, c_5, c_6] = [0, 5, 95, 5, 100, 0]$.

Similarly to Section 4, the optimal policy can be computed by placing Eq.7 into Eq.8. It should be noted that using different MIP algorithms might lead to different solutions Δ, because the size of the optimal set of policy-switching states is usually less than N, and so Δ might contain some meaningless information. For example, if phase k does not exist ($\forall i, a \; x_{ia}^k = 0$), then any $\Delta^k = \{\Delta_a^k\}$ satisfying $\sum_a u_a \Delta_a^k \leq \omega$ is possible. Nevertheless, employing different MIP algorithms cannot change the optimal value of the objective function since each MIP algorithm attempts to maximize the same objective function while satisfying the same list of constraints. The solution using the *cplex* MIP solver is

$$[\Psi_1, \Psi_2, \Psi_3, \Psi_4, \Psi_5, \Psi_6] = [1, 0, 0, 1, 0, 0]$$

$$\begin{vmatrix} \Delta_1^1, \Delta_2^1, \Delta_3^1, \Delta_4^1, \Delta_5^1, \Delta_6^1 \\ \Delta_1^2, \Delta_2^2, \Delta_3^2, \Delta_4^2, \Delta_5^2, \Delta_6^2 \\ \Delta_1^3, \Delta_2^3, \Delta_3^3, \Delta_4^3, \Delta_5^3, \Delta_6^3 \\ \Delta_1^4, \Delta_2^4, \Delta_3^4, \Delta_4^4, \Delta_5^4, \Delta_6^4 \\ \Delta_1^5, \Delta_2^5, \Delta_3^5, \Delta_4^5, \Delta_5^5, \Delta_6^5 \\ \Delta_1^6, \Delta_2^6, \Delta_3^6, \Delta_4^6, \Delta_5^6, \Delta_6^6 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ - & - & - & - & - & - \\ - & - & - & - & - & - \\ 1 & 1 & 0 & 0 & 0 & 0 \\ - & - & - & - & - & - \\ - & - & - & - & - & - \end{vmatrix}$$

where '-' indicates meaningless information.

So, the optimal policy in the phase initiated at S_1 is $[S_1 \rightarrow a_2, S_2 \rightarrow noop, S_3 \rightarrow noop, S_5 \rightarrow noop]$ and the optimal policy in the phase initiated at S_4 is $[S_2 \rightarrow noop, S_3 \rightarrow noop, S_4 \rightarrow noop, S_5 \rightarrow a_6]$. The value of the objective function is 96.96. In comparison with the total expected reward computed in Section 4, the agent receives a lower reward because it cannot afford the cost of switching policies at S_3 and S_5.

6. EMPIRICAL RESULTS

To this point, we have described a series of increasingly difficult MPPs and techniques for solving them, using a simple example to illustrate these ideas. Ultimately, the significance of these techniques hinges on their ability to automate the phasing of increasingly difficult problems, where difficulty can arise along various dimensions. In this section, we give a preliminary empirical evaluation of our techniques focusing on problems with a larger and more interconnected state space, and considering various degrees of agent constrainedness.

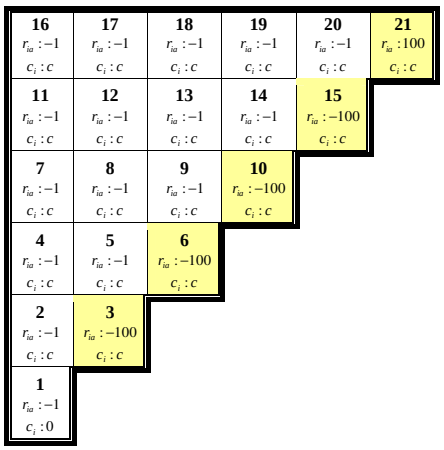

Figure 3: Experimental setup.

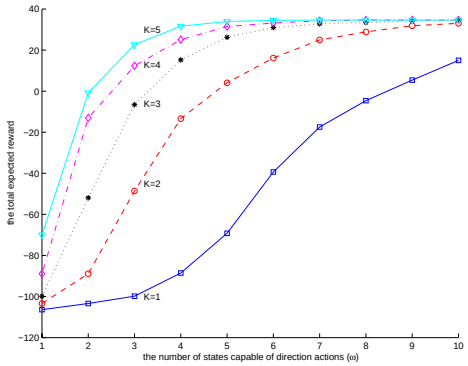

Figure 4: Experimental results ($tlim = \infty$) for a fixed number K of policy-switching states, for different values of K.

Suppose that an agent is situated in the environment shown in Figure 3. Beginning in the START state S_1, it must choose an action in the space $\{noop, Up, Left, Down, Right\}$ at each time step. The mission terminates when the agent reaches one of the GOAL states $\{S_3, S_6, S_{10}, S_{15}, S_{21}\}$. The system is stochastic. *noop* randomly moves the agent to one of the neighboring states (each direction has probability 0.25). Each direction action in the set $\{Up, Left, Down, Right\}$ achieves the intended effect with probability 0.75, moves the agent in either direction perpendicular to the intended direction with probability 0.1, and goes in the opposite direction with probability 0.05. Furthermore, if the agent bumps into a wall, it stays at the same state. The architecture of the agent is constrained – it can only implement direction actions in a limited number of states. That is, for some states, the agent might not be able to implement direction actions, in which case *noop* is executed. The reward r_{ia} and the cost c_i are shown in Figure 3.

Figure 4 shows the optimal total expected rewards by assuming that the number of policy-switching states K is fixed (where START state S_1 is always a policy-switching state). We can see that breaking the mission into multiple phases can significantly improve the total expected reward of the

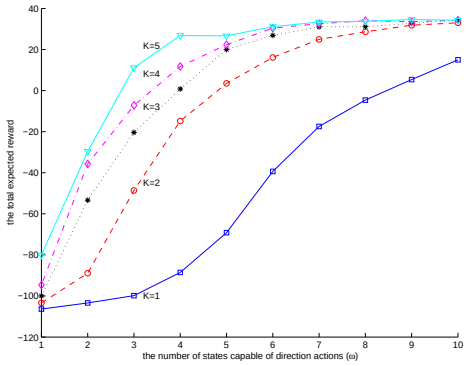

Figure 5: Experimental results ($tlim = 100$) for a fixed number of K policy-switching states, for different values of K.

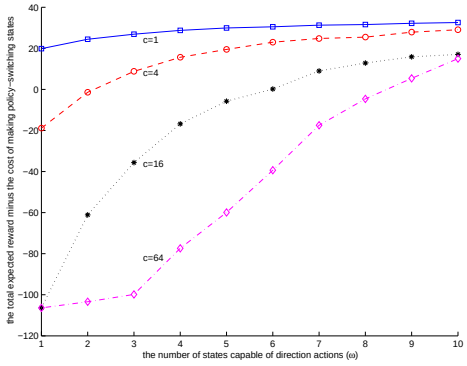

Figure 6: Experimental results ($tlim = \infty$) for incorporating cost c for policy switching, for different values of c.

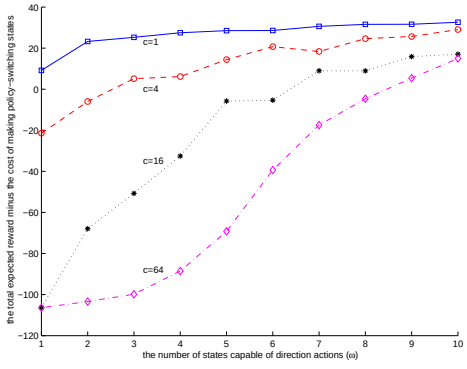

Figure 7: Experimental results ($tlim = 100s$) for incorporating cost c for policy switching, for different values of c.

constrained agent, especially when constraints are tightened.

It should be noted that mixed integer programming solvers become slower in finding the optimal solution as the number of integer variables increases. That is, when the number of states or the number of phases is large, the MIP solvers might need a long time to compute optimal policies. However, state-of-the-art MIP solvers (such as *cplex*) are usually able to return a *good* solution using much less time. Figure 5 shows the total expected rewards by limiting the computational time *tlim* to 100 seconds for each data point.

Experimental results for incorporating cost for policy switching are shown in Figure 6 ($tlim = \infty$) and Figure 7 ($tlim = 100s$), where the number of policy-switching states is unlimited, and c is the cost for making a state (except S_1) into one that is conducive to policy switching. These results illustrate that, given the degree of constrainedness, our algorithms can wisely determine policy-switching states based upon the cost c so that the total expected reward of the agent minus the cost for making policy-switching states is maximized.

7. SUMMARY AND FUTURE WORK

In this paper, we present algorithms for automating the process of finding and using mission phases for constrained agents. We analyze several variations of this problem that correspond to different classes of important constrained-agent problems, and show through analysis and experiments that our techniques can increase an agent's rewards for varying levels of constraints on the agent and on the phases.

In the future, we plan to work on efficient approximate algorithms since mixed integer programming is NP complete, which significantly slows down when the number of states or the number of phases becomes large. We also intend to study the variance caused by deterministic/randomized choices of phase policies at policy-switching states. Finally, we aim to extend our algorithms to multi-agent environments. In many multi-agent systems, the mission is divided into multiple phases. At the end of each phase, agents converge to accomplish some joint task or to coordinate strategies to be used in the next phase. Designing a multi-agent mission phasing algorithm is thus also a promising future direction.

8. ACKNOWLEDGEMENTS

This material is based upon work supported in part by Honeywell International and by the DARPA/IPTO COORDINATORs program and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

The authors thank Dmitri Dolgov and three anonymous reviewers for their helpful suggestions and comments.

9. REFERENCES

[1] E. Altman. Constrained Markov decision processes with total cost criteria: Lagrange approach and dual LP. *Methods and Models in Operations Research*, 48:387–417, 1998.

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

[3] A. Bockmayr and Y. Dimopolous. Mixed integer programming models for planning problems. In *CP'98 Workshop on Constraint Problem Reformulation*, 1998.

[4] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley & Sons, New York, 1998.

[5] D. A. Dolgov and E. H. Durfee. Constructing optimal policies for agents with constrained architectures. Technical Report CSE-TR-476-03, University of Michigan, 2003.

[6] D. A. Dolgov and E. H. Durfee. Constructing optimal policies for agents with constrained architectures (abstract). In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 974–975, 2003.

[7] E. Feinberg. Constrained discounted Markov decision processes and Hamiltonian cycles. *Mathematics of Operations Research*, 25:130–140, 2000.

[8] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 220–229, 1998.

[9] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

[10] L. Kallenberg. Linear programming and finite Markovian control problems. *Mathematisch Centrum, Amsterdam*, 1983.

[11] H. Kautz and J. Walser. Integer optimization models of AI planning problems. *Knowledge Engineering Review*, 15(1):101–117, 2000.

[12] D. J. Musliner, E. H. Durfee, and K. G. Shin. CIRCA: A cooperative intelligent real time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1561–1574, 1993.

[13] D. J. Musliner, E. H. Durfee, and K. G. Shin. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence*, 74(1):83–127, 1995.

[14] R. Parr. Flexible decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 422–430, 1998.

[15] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP 01)*, pages 37–48, 2001.

[16] M. L. Puterman. *Markov Decision Processes*. John Wiley & Sons, New York, 1994.

[17] P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *Proceedings of National Conference on Artificial Intelligence*, pages 585–590, 1999.

[18] T. Vossen, M. Ball, A. Lotem, and D. Nau. On the use of integer programming models in AI planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Stockholm, Sweden*, pages 304–309, 1999.

[19] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, 1998.