Solving Very Large Weakly Coupled Markov Decision Processes

Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin Leslie Pack Kaelbling, Thomas Dean*
Computer Science Department, Box 1910
Brown University, Providence, RI 02912-1210
{nm, milos, kek. ldp, lpk, tld}@cs.brown.edu

Craig Boutilier†
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, Canada
cebly@cs.ubc.ca

Abstract

We present a technique for computing approximately optimal solutions to stochastic resource allocation problems modeled as Markov decision processes (MDPs). We exploit two key properties to avoid explicitly enumerating the very large state and action spaces associated with these problems. First, the problems are composed of multiple tasks whose utilities are independent. Second, the actions taken with respect to (or resources allocated to) a task do not influence the status of any other task. We can therefore view each task as an MDP. However, these MDPs are weakly coupled by resource constraints: actions selected for one MDP restrict the actions available to others. We describe heuristic techniques for dealing with several classes of constraints that use the solutions for individual MDPs to construct an approximate global solution. We demonstrate this technique on problems involving thousands of tasks, approximating the solution to problems that are far beyond the reach of standard methods.

1 Introduction

Markov decision processes [12, 16] have proven tremendously useful as models of stochastic planning and decision problems. However, the computational difficulty of applying classic dynamic programming algorithms to realistic problems has spurred much research into techniques to deal with large state and action spaces. These include function approximation [4], reachability considerations [8] and aggregation techniques [11, 6, 7].

One general method for tackling large MDPs is *decomposition* [10, 15, 17, 5]. An MDP is either specified in terms of a set of "pseudo-independent" subprocesses [17] or automatically decomposed into such subprocesses [5]. These subMDPs are then solved and the solutions to these subMDPs are merged, or used to construct an approximate global solution. These techniques can be divided into two broad classes: those in which the state space of the MDP is divided into regions to form subMDPs, so that the MDP is the union (in a loose sense) of the subMDPs [10, 15]; and those in which the subMDPs are treated as concurrent processes, with their (loosely) cross-product forming the global MDP. In this paper, we focus on the latter form of decomposition: it offers great promise by allowing one to solve subMDPs that are exponentially smaller than the global MDP. If these solutions can be pieced together effectively, or used to guide the search for a global solution directly, dramatic improvements in the overall solution time can be obtained.

The problems we address here are sequential stochastic resource allocation problems. A number of different tasks, or objectives, must be addressed and actions consist of assigning various resources at different times to each of these tasks. We assume that each of these tasks is *additive utility independent* [13]: the utility of achieving any collection of tasks is the sum of rewards associated with each task. In addition, we assume that state space of the MDP is formed from a number of features that, apart from resources, are relevant only to a specific task. Furthermore, an assignment of resources to one task has no bearing on the features relevant to any other task. This means that each task can be viewed as an independent subprocess whose rewards and transitions are independent of the others, given a fixed action or policy (assignment of resources). [1]

Even this degree of independence, however, does not generally make it easy to find an optimal policy. Resources are usually constrained, so the allocation of resources to one task at a given point in time restricts the actions available for others at every point in time. Thus, a complex optimization problem remains. If there are no resource constraints, the processes are completely independent. They can be solved individually, and an optimal global solution determined by concurrent execution of the optimal local policies; solution time is determined by the size of the subMDPs. With resource constraints, local optimal solutions can be computed, but merging them is now non-trivial. The question of how best to exploit local solutions to determine a global policy is the subject of

*This work was supported in part by DARPA/Rome Labs Planning Initiative grant F30602-95-1-0020 and in part by NSF grants IRI-9453383 and IRI-9312395

†This work was supported by NSERC Research Grant OGP0121843 and IRIS-II Project IC-7, and was undertaken while the author was visiting Brown University. Thanks also to the generous support of the Killam Foundation.

[1]This model can be applied more generally to processes where the action can be broken into several components, each affecting a different process independently; resource allocation is a specific example of this.

this paper. We note that in resource allocation problems, the action space is extremely large (every possible assignment of resources to tasks), making other standard approximation methods such as neurodynamic programming [4] unsuitable.

Singh and Cohn [17] treat a version of this problem, in which there are constraints on the feasible joint action choices. As they observe, the value functions produced in solving the subMDPs can be used to obtain upper and lower bounds on the global value function. These bounds are used to improve the convergence of value iteration (via a form of *action elimination* [16]) for the global MDP. Unfortunately, their algorithm requires explicit state-space and action-space enumeration, rendering it impractical for all but moderate-sized MDPs.

We take a different approach in this paper: we are willing to sacrifice optimality (assured in Singh and Cohn's algorithm) for computational feasibility. To do this, we develop a several greedy techniques to deal with variants of this problem (in which the types of resource constraints differ). A hallmark of these heuristic algorithms is their division into two phases. An *off-line phase* computes the optimal solutions and value functions for the subMDPs associated with individual tasks. In an *on-line* phase, these value functions are used to compute a gradient for a heuristic search to assign resources to each task based on the current state. Once an action is taken, these resource assignments are reconsidered in light of the new state entered by the system.

This problem formulation was motivated by a military air campaign planning problem in which the tasks correspond to targets, and in which there are global constraints on the total number of weapons available as well as instantaneous constraints (induced by the number of available aircraft) on the number of weapons that may be deployed on any single time step. Actions have inherently stochastic outcomes and the problem is fully observable. This type of problem structure is fairly general, though, and can also be seen in domains such as stochastic job shop scheduling, allocation of repair crews to different jobs, disaster relief scheduling, and a wide variety of bandit-type problems [3].

We are able to solve problems of this type involving hundreds of tasks (with a state space exponential in this number) and thousands of resources (with an action space factorial in this number). Such problems are far beyond the reach of classic dynamic programming techniques and typical approximation methods such as neurodynamic programming.

In Section 2 we discuss some relevant background on MDPs and define our specific problem class formally. In Section 3, we describe *Markov task decomposition* (MTD) as a means of breaking down large, loosely coupled decision processes and describe, in very general terms, how solutions for the subMDPs might be used to construct a global solution in the presence of various types of action constraints. In Section 4, we describe the air campaign planning problem in some detail, and show how particular characteristics of this problem make it especially well-suited to both MTD and our heuristic policy construction methods. Section 5 demonstrates the results of MTD applied to very large instances of this problem. We conclude in Section 6 with some remarks on the reasons for MTD's success and future work.

2 Markov Task Sets

A (finite) *Markov decision process* is a tuple $\langle S, A, T, R \rangle$ where: S is a finite set of states; A is a finite set of actions; T is a transition distribution $T : S \times A \times S \to [0, 1]$, such that $T(s, a, \cdot)$ is a probability distribution over S for any $s \in S$ and $a \in A$; and $R : S \times A \times S \to \mathbb{R}$ is a bounded reward function. Intuitively, $T(s, a, w)$ denotes the probability of moving to state w when action a is performed at state s, while $R(s, a, w)$ denotes the immediate utility associated with the transition $s \rightsquigarrow w$ under action a. More generally, both the probabilities and rewards maybe *non-stationary*, depending also on the time at which the action is performed or the transition is made.

Given an MDP, the objective is to construct a *policy* that maximizes expected accumulated reward over some horizon of interest. We focus on *finite horizon* decision problems. Let H be our horizon: the aim is construct a non-stationary policy $\pi = \langle \pi^0, \cdots, \pi^{H-1} \rangle$, where $\pi^t : S \to A$, whose *value*

$$V_\pi^H(s) = E(\sum_{t=0}^{H-1} R(s^t, \pi^t(s^t), s^{t+1}))$$

is maximum. Standard dynamic-programming methods [2, 16] can be used to compute a sequence of optimal k-stage-to-go value functions up to the horizon of interest, from which an optimal policy can be derived.

We consider a special form of MDP suitable for modeling the stochastic resource allocation problems described in the introduction. A *Markov task set* (MTS) of n tasks is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, c, M_g, M_r \rangle$, where

- $\mathcal{S}$ is a vector of state spaces, $S_1, \ldots, S_n$, where each S_i is the set of primitive states of Markov task i;
- $\mathcal{A}$ is a vector of action spaces, $A_1, \ldots, A_n$, where each A_i is a set of integers from 0 to some limit, describing the allocation of an amount of resource to task i;[2]
- $\mathcal{R}$ is a vector of reward functions $R_1, \ldots, R_n$, where $R_i : S_i \times A_i \times S_i \times \text{Time} \to \mathbb{R}$, specifying the reward conditional on the starting state, resulting state and action at each time;[3]
- $\mathcal{T}$ is a vector of state transition distributions, $T_1, \ldots, T_n$, where $T_i : S_i \times A_i \times S_i \to [0, 1]$, specifying the probability of a task entering a state given the previous state of the task and the action;
- c is the cost for using a single unit of the resource;
- M_l is the instantaneous (local) resource constraint on the amount of resource that may be used on a single step;
- M_g is the global resource constraint on the amount of the resource that may be used in total.

We again assume a finite horizon H.[4] An MTS induces an MDP in the obvious way: the state space consists of the

[2] It may be possible to extend this work to apply to real-valued amounts of resources and to multiple resource types.

[3] If reward values are stationary the time index may be omitted.

[4] Our techniques may be extended to other optimality criteria, such as infinite-horizon discounted or average reward.

cross product of the individual state spaces and the available resources; the action space is the set of resource assignments, with an assignment being feasible at a state only if its sum exceeds neither M_l nor the total resources available at that state; rewards are determined by summing the original component rewards and action costs; and transition probabilities are given by multiplying the (independent) individual task probabilities (with the change in resources being determined by the action).

Instead of formulating this "flat" MDP explicitly, we retain the factored form as much as possible. The goal, then, is to find an optimal non-stationary policy $\pi^* = \langle \pi^0, \ldots, \pi^{H-1} \rangle$, where $\pi^t = \langle \pi_1^t, \cdots, \pi_n^t \rangle$ and each $\pi_i^t : S \rightarrow A_i$ is a *local policy* for task i, that maximizes

$$E\left[\sum_{t=0}^{H-1}\sum_{i=1}^{n} T(s_i^t, \pi_i^t(s^t), s_i^{t+1}) R_i(s_i^t, \pi_i^t(s^t), t) - c \cdot \pi_i^t(s^t)\right]$$

subject to the constraints

$$\forall t < H, \sum_{i=1}^{n} \pi_i^t(s^t) \leq M_l \quad \text{and} \quad \sum_{t=0}^{H-1}\sum_{i=1}^{n} \pi_i^t(s^t) \leq M_g$$

For simplicity, we have described MTSs involving only a single resource type. In general, there may be multiple resources, each of which has a cost and may be subject to local and global constraints. In Section 4 we present a problem with two resources with interrelated constraints. Note that MTSs allow one to model both reusable resources, such as machines on a shop floor (with local but no global constraints,) and consumable resources, such as raw materials (that have global and possibly induced local constraints).

Finding an optimal policy is a very hard problem even for small MTSs, because the equivalent MDP is very large. It is, for all practical purposes, impossible to solve exactly unless the number of tasks, the individual state spaces and the available resources are very small. The major source of difficulty is that the decision to apply a resource to a given task influences the availability of that resource (either now or in the future) for other tasks. Thus, the tasks, while exhibiting tremendous independence, still have strongly interacting solutions. A "local policy" for each task must take into account the state of each of the other tasks, precluding any state-space reduction in general[5]. We now turn our attention to approximation strategies that limit the scope of these interactions.

3 Markov Task Decomposition

Our approximation strategy for MTSs is called *Markov task decomposition* (MTD).

The MTD method is divided into two phases. In the first, *off-line phase*, value functions are calculated for the individual tasks using dynamic programming. In the second, *on-line phase*, these value functions are used to calculate the next action as a function of the current state of all processes.

[5]If there are no resource constraint, the sub-processes can be solved individually and the local policies can be defined as mappings $\pi_i^t : S_i \rightarrow A_i$.

3.1 Global Constraints Only

We will first consider the case in which there is only a global resource constraint, but no limit on the total number of resources applied on a single step.

In the off-line phase, we compute the *component value functions* $V_i(s_i, t, m)$ where s_i is the current state of task i, $0 \leq t \leq H$ is the time step, and m is the number of resources remaining:

$$V_i(s_i, t, m) = \max_{a \leq m} \sum_{s_i' \in S_i} T_i(s_i, a, s_i')[R_i(s_i, a, s_i', t) + V_i(s_i', t+1, m-a)] - c \cdot a \quad (1)$$

where $V_i(s_i, H, m) = 0$. This is the expected cumulative reward of the optimal policy for task i starting in state s_i at time t using at most m resources. In other words, if we ignored all other tasks and had m resources to allocate to task i, we would expect this value. It is useful to note that, even at the last stage, it may be suboptimal to spend all (or even any) of the remaining resources.

It is relatively simple to compute V_i using dynamic programming as long as we have some way of tightly bounding the values of m and t that must be considered. In Section 4, we describe a domain for which tight bounds on these quantities are available.

With these V_i in hand, we proceed to the on-line phase. We are faced with a particular situation, described by the current state $s = \langle s_1, \ldots, s_n \rangle$, remaining global resources m_g, and time-step t. We must calculate $a = \langle a_1, \ldots, a_n \rangle$, the action to be taken (i.e., the resources to be spent) at the current time step (where a_i is applied to task i). Since we are ignoring instantaneous constraints, we require only that $\sum_i a_i \leq m_g$. However, allocating all resources at the current time step will generally not be optimal. Optimal allocation would be required to take into account future contingencies, their probabilities and the value of holding back resources for these future contingencies.

Rather than solve this complex optimization problem, we rely on the fact that the local value functions V_i give us some idea of the value of assigning $m_i \leq m_g$ resources to task i at time t. Furthermore, V_i implicitly determines a policy for task i, telling us how many of the resources $a_i \leq m_i$ should be used at the current step t. Thus, MTD works in the following loop, executed once per decision stage:

(a) Using the functions $V_i(s_i, t, \cdot)$, heuristically assign resources m_i to task i such that $\sum_{i \leq n} m_i \leq m_g$.

(b) Use V_i and m_i to determine a_i, the action to be taken currently w.r.t. task i; that is

$$a_i = \arg\max_{a \leq m_i} \sum_{s_i' \in S_i} T_i(s_i, a, s_i')[R_i(s_i, a, s_i', t) + V_i(s_i', t+1, m_i - a)] - c \cdot a$$

(c) Execute action $a = \langle a_1, \cdots, a_n \rangle$, observe resulting state $s = \langle s_1, \cdots, s_n \rangle$, and compute remaining resources $m_g' = m_g - \sum_{i \leq n} a_i$.

The one component of MTD that has been left open is the heuristic allocation of resources to tasks. Doing this well will generally require specific domain knowledge. We describe a greedy approach in Section 4 below that works extremely well in the air campaign planning domain, though the fundamental characteristics of this problem hold true of a wide class of problem domains.

This approach is plausible, but even if the m_i are chosen optimally with respect to the criteria described above, the policy produced will generally be suboptimal for the following reasons.[6] We estimate the utility of an allocation m_i to task i using V_i, which is exactly the utility of solving task i with m_i resources; clearly,

$$\hat{V}(s,t,m) = \sum_{i=1}^{n} V_i(s_i,t,m_i)$$

is a lower bound on V^*; it is the value we would achieve if we made the allocations at step t and never re-evaluated them. In particular, given m_i resources for task i, MTD allocates a_i resources to the task at the current stage based only on V_i. The optimal Bellman equations indicate that an optimal allocation a_i must not only take into account the future course of task i, but also reason about future contingencies regarding other tasks, and assess the value of reallocating some of these resources to other tasks in the future.

3.2 Adding Instantaneous Constraints

It is quite difficult to incorporate local constraints in a satisfying way. An obvious strategy is to simply enforce the constraint that $\sum_{i=1}^{n} a_i \leq M_l$ to the on-line phase of MTD. This will result in the generation of admissible policies, but they may be of poor quality—$\hat{V}$ is likely to be a serious overestimate of the value function. This is because the allocations m_i determined by the V_i in step (a) above may be based on the assumption that more than M_l resources can be used in parallel.

Despite the potential drawbacks, we pursue a strategy of this type in the application described in Section 4. This type of strategy has the appeal of computational simplicity and leads to reasonably good results in our domain. However, more complex strategies for dealing with instantaneous constraints can easily be accommodated within the MTD framework. Such strategies will be the subject of future study.

4 Example: Air Campaign Planning

In the simplified air campaign planning problem, tasks correspond to targets and there are two resource types: weapons and planes. The status of a target is either damaged or undamaged: $S_i = \{d, u\}$. Each target has a window of availability $[t_i^s, t_i^e]$, whose length is denoted $w_i = t_i^e - t_i^s + 1$, and a reward r_i; if it is damaged during the window, then the reward r_i is received:

$$R_i(s_i, a_i, s_i', t) = \begin{cases} r_i & \text{if } t_i^s \leq t \leq t_i^e \text{ and } s_i = u \text{ and } s_i' = d \\ 0 & \text{otherwise} \end{cases}$$

With probability p_i, a single weapon will damage target i; a "noisy-or" model is assumed for multiple weapons, in which a single "hit" is sufficient to damage the target and individual weapons' hit probabilities are independent. That is,

$$T_i(s_i, a_i, s_i') = \begin{cases} 0 & \text{if } s_i = d \text{ and } s_i' = u \\ 1 & \text{if } s_i = d \text{ and } s_i' = d \\ q_i^{a_i} & \text{if } s_i = u \text{ and } s_i' = u \\ 1 - q_i^{a_i} & \text{if } s_i = u \text{ and } s_i' = d \end{cases}$$

where $q_i = 1 - p_i$ is the probability of a weapon missing the target. There is a cost, c, per weapon used.[7] Each plane has a capacity K_p (which we take to be fixed for simplicity) and can carry only up to K_p weapons. A plane loaded with a weapons and assigned to a target will deliver all a weapons. We will consider a variety of different constraints. In Section 4.3 we treat the case in which the only constraint is a global constraint on the total number of weapons, M_g. In Section 4.4 we treat the case in which the only constraint is a local constraint on the number of planes, M_l, that can be used at a single stage. Since each plane can only carry a limited number of weapons, any constraint on planes induces a constraint on weapons. This is a more sophisticated type of local constraint than previously. Now, each action must satisfy:

$$\sum_{i=1}^{n} \lceil a_i / K_p \rceil \leq M_l$$

Finally, we combine global weapon constraints with local plane constraints in Section 4.5.

4.1 Calculating Component Value Functions

The following discussion is somewhat brief and informal; a formal discussion with detailed derivations will be provided in a forthcoming technical report.

Within this problem, the component value functions can be considerably simplified. First, since every state in which the target has been damaged has value 0, we need only compute value functions for the state in which the target is undamaged. Second, since there is only a restricted window of opportunity for each target i, we need only compute the value function for a horizon equal to the number of time steps w_i in the target's window of opportunity. Since window lengths are typically much shorter than the horizon for the entire problem, this considerably simplifies the dynamic programming problem: we need only compute $V_i(u, m, t)$ for $t_i^s \leq t \leq t_i^e$, and then we apply:

$$V_i(u,m,t) = \begin{cases} V_i(u,m,t_i^s) & \text{if } t < t_i^s \\ 0 & \text{if } t > t_i^e \end{cases}$$

Another factor that strongly influences the "shape" of the local value functions (and ultimately, our heuristic algorithm) is the noisy-or transition model. Because of this, the probability of damaging the target of any policy that uses m weapons in no more than g steps, depends only on m (not on when the weapons are used) and is equal to $1 - q_i^m$. Policies may differ in expected utility, however, depending on how they choose to allocate weapons over time, which affects the expected number of weapons used. If $w_i \geq m$ then it is never optimal to send more than one weapon at a time. Otherwise,

[6] Note that the policy produced by MTD is constructed incrementally; indeed, it isn't a policy *per se* since it only plans for states that are actually reached.

[7] We ignore the cost per plane in the present paper.

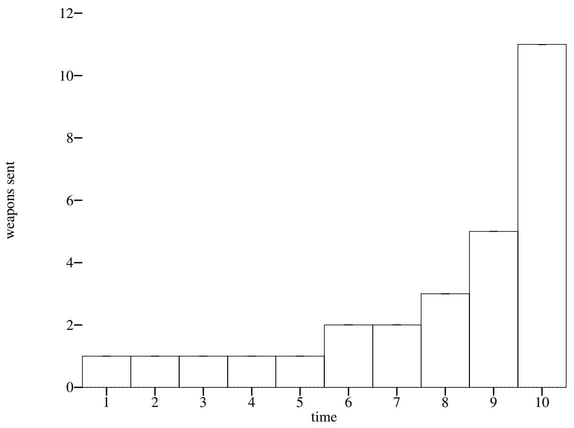

Figure 1: An instance of optimal policy for single-target problem: number of weapon sent if the target is still undamaged at time t, as a function of t ($p_i = 0.25$, $r_i = 90$, $w_i = 10$ and $c = 1$).

the optimal policy sends an increasing number of weapons at each step, in the case that the weapons on the previous step failed to damage the target.

Figure 1 shows an example of such a single target policy with a given window, reward and hit probability (it is optimal for any allocation of weapons greater than the cumulative total shown).

Furthermore, we can show that $V_i(u, m, t)$ increases monotonically with m until a point $m^*_{i,t}$, at which point it remains constant: $m^*_{i,t}$ is the point at which marginal utility of resource allocation becomes zero, and where the marginal utility of resource use is negative (the cost of a weapon exceeds the value of the small increase in success probability, so even if it is allocated, it will not be used). This implies that we need only compute $V_i(u, m, t)$ for $m \leq m^*_{i,t}$, again significantly reducing the effort needed to compute V_i.

For each target i, each $t \in [t^s_i, t^e_i]$, and each $m \leq m^*_{i,t}$, we will compute $V_i(u, m, t)$ and store these results in a table to be used in the on-line phase of MTD. We can do this using the dynamic programming equation

$$V_i(u, m, t) = \max_{0 \leq a \leq m} \left[(1 - q^a_i) r_i - c_w a + q^a_i V_i(u, m - a, t + 1) \right],$$

where $V_i(u, m, t^e_i + 1) = V_i(u, 0, t) = 0$. The value of spending b' weapons can be described using three terms: the first is the expected reward due to damaging the target on the current time step, the second is the cost of using a weapons, and the third is the future value of trying to damage the target with $m - a$ weapons left.

4.2 No Resource Constraints

If there are no resource constraints, the on-line phase of MTD is not required. The tasks are completely decoupled and the optimal policy π^t is described by the component value functions (recall that $\pi^t_i(d) = 0$; no action is required for a damaged target):

$$\pi^t_i(u) = \arg\max_{a_i} (1 - q^{a_i}_i) r_i - c_w a_i + q^{a_i}_i V_i(u, m^*_{i,t+1}, t+1) \quad (2)$$

(for any t within i's window). This requires a simple search over values of a_i, bounded by $a_i \leq m^*_{i,t}$.

4.3 Weapon (Global) Constraints Only

With constraints on the number of weapons available, the on-line phase is crucial. We have the component value functions V_i at our disposal, and are given the current state s of all targets, the number of weapons remaining m_g, and the time t. Our goal is to choose m_i—the weapons to assign to each target i with state $s_i = u$ and such that $t \leq t^f_i$ —according to step (a) of the on-line algorithm in Section 3.1; that is, to maximize $\sum V_i(u, m_i, t)$.

To do this, we adopt a greedy strategy. Define

$$\Delta V_i(u, m, t) = V_i(u, m + 1, t) - V_i(u, m, t) \quad (3)$$

to be the marginal utility of assigning an additional weapon to target i, given that m weapons have already been assigned to it. We assign weapons one by one to the target that has the highest value $\Delta V_i(u, m, t)$ given its current assignment of weapons (i.e., gradient ascent on $\sum V_i$). This proceeds until all m_g weapons have been assigned or $\Delta V_i(u, m, t) \leq 0$ for all i. The concavity of the local value functions assures:

Proposition 1 *The process described above chooses b_i to maximize $\sum V_i(u, m_i, t)$.*

Despite this, as we argued above, this does not necessarily result in an optimal policy. However, in this domain, the empirical results are impressive, as we discuss in Section 5.

4.4 Plane (Instantaneous) Constraints Only

Even with an unlimited number of weapons (or as many as required to reach zero marginal utility), we generally have to deal with constraints on the number of simultaneously deliverable weapons (i.e., number of planes available). The strategy we adopt is similar to the one above, except we greedily allocate planes instead of weapons. The one subtlety lies in the fact that it may not be optimal to load a plane to capacity (recall, that all weapons on a plane are delivered).

We proceed at time t by allocating planes one by one to active targets. We assume (optimistically) that all targets in the future can be allocated their optimal number of weapons (this is optimistic because of future plane constraints, not because of weapon availability); in other words, for computational reasons, we deal with plane constraints only at the current stage. Assume we have assigned n_i planes and $a_i \leq n_i \cdot K_p$ weapons to target i so far. For each active target i, we compute a'_i, the number of bombs that the new plane would carry:

$$a'_i = \min \left\{ K_p, \pi^t_i(u) - K_p \cdot n_i \right\}$$

where $\pi^t_i(u)$ is given by (2). This can be used to compute the marginal expected utility of assigning a new plane to any active target:

$$\Delta V_i = q^{a_i}_i \left(1 - q^{a'_i}_i\right) r_i - c_w a'_i + q^{a_i}_i \left(q^{a'_i}_i - 1\right) V_i(u, m^*_{i,t+1}, t + 1).$$

As in the case of global constraints only, we assign planes to active targets greedily until $\sum n_i = M_l$ or $\Delta V_i \leq 0$ for all i. Note that marginal utility is associated with increasing the number of planes, not weapons as in the previous section.

4.5 Weapon and Plane Constraints

Our approach in this case will necessarily be more complicated. We begin by assigning weapons to targets using the greedy strategy outlined in Section 4.3; no plane constraints are accounted for. The result is an assignment of resources m_i to each target. An action $a = \langle a_1, \cdots, a_n \rangle$ is determined for the current stage, and we assign n_i planes to each target at the current stage that will suffice to carry the a_i weapons. This action may be infeasible however if $\sum n_i > M_l$ (more planes are required to carry out action a than are available). We thus begin a greedy deallocation-reallocation process.

Deallocation requires that we remove certain weapons from the current assignment a. We do this by greedily removing the assigned planes one by one until $\sum n_i = M_l$ (note that i need only range over active targets). Intuitively, we proceed as follows: first we compute the number of weapons a'_i that would be removed from target i if we deallocated a single plane; we compute the change in utility that would accompany this deallocation if we were to "optimally" reallocate these weapons to a new target (or possibly the same target, but forced to be used at a later stage); and then we deallocate the plane and perform the reallocation that results in the smallest decrease in utility. However, we will see that this requires some care.

At any point in time, we have a list of (active) targets which have had planes deallocated. For any such target i, we may consider assigning new weapons to it that have been deallocated from some target j. But since we do not want to consider providing a new plane for i at the *current* stage (one has just been taken away), we compute ΔV_i, the marginal utility of adding a weapon to i, as follows:

$$\Delta V_i = q_i^{a_i} \left[V_i(m_i - a_i + 1, t+1) - V_i(m_i - a_i, t+1) \right] \quad (4)$$

That is, we consider that this weapon must be used at some time *after* the current stage. For any target that has not had a plane deallocated, ΔV_i is computed as in (3). We let $\Delta_n V_i$ denote the change in utility if we assign n new weapons (instead of 1).

Let $a'_i = a_i - (m_i - 1)K_p$ be the number of weapons that will be removed from (active) target i if one of its n_i planes is deallocated to satisfy the instantaneous constraint. Then compute δV_i, the value of reallocating the a'_i weapons optimally: we do this by adding i to the deallocated list (temporarily) and simulating the greedy algorithm described in Section 4.3.[8] The only difference is that we use (4) as the measure of marginal utility for any target on the deallocated list. Notice that weapons taken from i can be reallocated to i, but the value of this reallocation is derived from using these at *later* stages. If d_j weapons are assigned to target j, where $\sum d_j = a'_i$, then $\delta V_i = \sum \Delta_{d_j} V_j$.

[8] By simulating, we mean that we compute the reallocation that would take place if we actually reallocated the weapons from i; this won't necessarily take place if we decide to deallocate a plane from some other target.

The quantities δV_i will be used to determine which target will have a plane deallocated. For any active target i, define

$$\begin{aligned} \nabla V_i \;=\; & q_i^{a_i - a'_i} \left(q_i^{a'_i} - 1 \right) r_i + c_w a'_i + \\ & q_i^{a_i - a'_i} \left(1 - q_i^{a'_i} \right) V_i(m_i - a_i, t+1) + \delta V_i \end{aligned}$$

This is the (negative) change in expected value by deallocating a'_i weapons (i.e., one plane) from target i.[9] We deallocate by choosing the target i with the largest ∇V_i. Once selected, a plane is deallocated from i, and the a'_i weapons are reallocated greedily. In fact, the d_j values used in the computation of δV_i can be stored and used for this purpose (the simulated reallocation can now be imposed).

We note that if any weapons are reallocated to an active target j, it may cause j to require an additional plane (in fact, this can occur for several active targets). To deal with this we simply allocate new planes. While the deallocation of one plane may cause the allocation of more planes, this process will eventually terminate, since no deallocated target can ever be reallocated weapons for *current* use.[10]

5 Empirical Results

To validate our heuristics, we tried them on several randomly-generated instances of the air campaign problem, and compared them to:

- the optimal policy calculated by flat DP
- the greedy policy that applies the action with highest expected immediate reward
- a "semi-greedy" policy that applies the action $\pi_i^t(u)$ given by (2) to each active target i, without regard to potential interactions

The calculation of the optimal policy by DP is infeasible for problems of moderate size. For instance, the solution time for a problem with 5 targets and 50 weapons is on the order of 10 minutes; for 6 targets and 60 weapons, up to 6 hours; and beyond that was not practically computable. In contrast, the execution time for MTD is shown in Figure 2. Without instantaneous constraints, MTD can solve a problem with thousands of targets and tens of thousands of weapons in under 15 minutes. A problem of 1000 targets, 10,000 weapons and 100 planes (imposing such constraints) can be solved in about 35 minutes.

We compare the quality of solutions produced by MTD with the optimal solutions produced by DP in Figure 3, though we are restricted to very small problems (5 tasks, with no instantaneous constraints; and 7 task, with instantaneous constraints) because of the computational demands of DP. We also compared the greedy and semi-greedy strategies.[11] The performance of MTD is encouraging, closely tracking optimal, though the performance of the greedy and semi-greedy

[9] Note that ∇V_i is not equal to $V_i(u, m_i - a'_i, t) - V_i(u, m_i, t)$.

[10] We are currently exploring more sophisticated strategies that prevent this from happening.

[11] For all but DP, whose solution has a known value, the results show average reward obtained over 1000 simulations of the process, as a function of M_g (initial global resources).

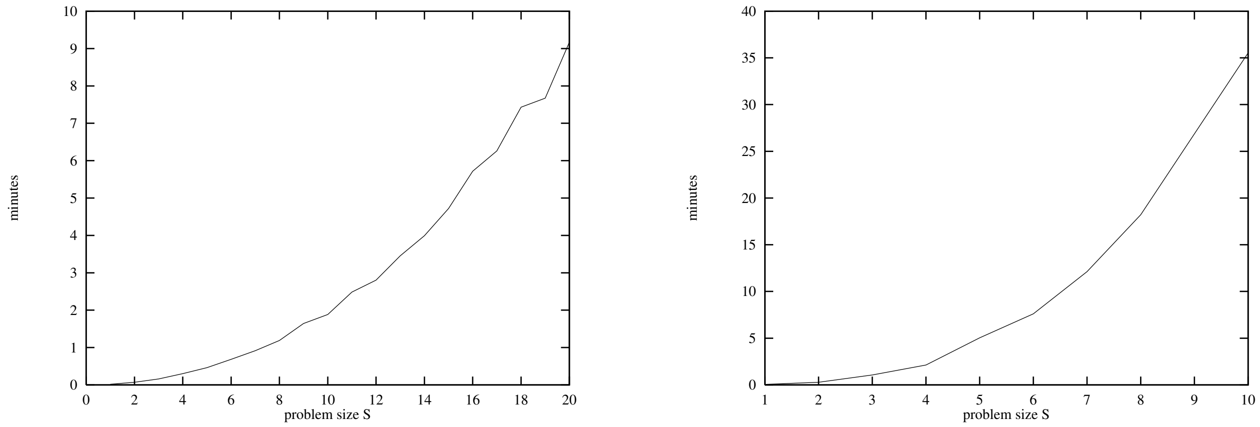

Figure 2: Complexity of MTD: time of execution as a function of the problem size S. A problem of size S has $100S$ targets, $1000S$ weapons and $10S$ planes. The graph on left shows the time to solve a problem with no local constraints. The graph on the right shows the time for a problem with local constraints (given by $10S$ planes).

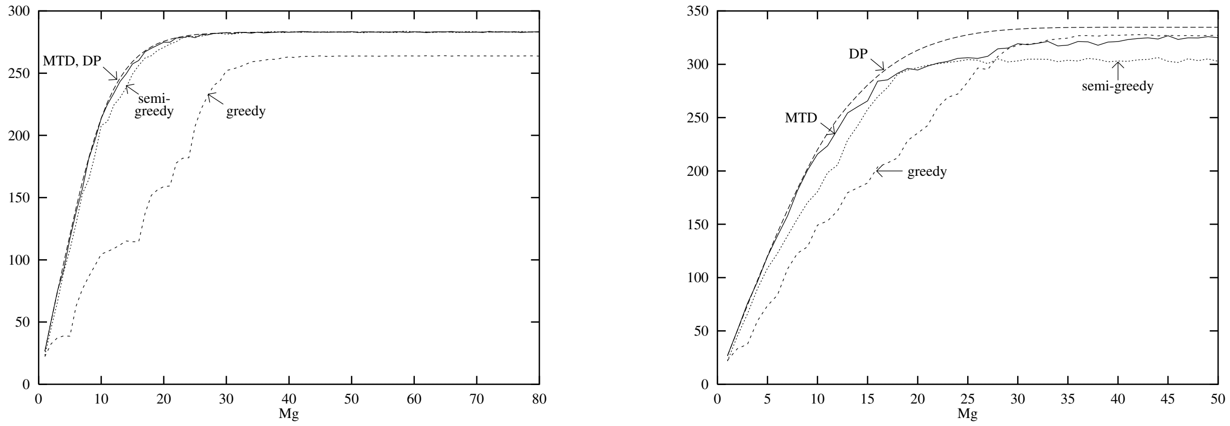

Figure 3: Comparison of the quality of policies generated by MTD, optimal DP, greedy and semi-greedy strategies on small test problems. The graph on left shows results for a 5-task problem with no local constraints. The graph on the right describes a 7-task problem with local constraints. Values are averaged over 1000 runs.

policies suggests these problems are not difficult enough to differentiate these from MTD.

On much larger problems, MTD compares much more favorably to both greedy methods, with Figure 4 showing their performance on 100 target (with instantaneous constraints) and 300 target (no constraints) problems. The policy produced by MTD performs substantially better than the greedy and semi-greedy policy. Such problems are well beyond the reach of classic DP.

6 Conclusions

We have presented the method of Markov task decomposition for solving large weakly coupled MDPs, in particular, stochastic resource allocation problems. We described several instantiations of this technique for dealing with different forms of resource or action constraints. The empirical results for the air campaign problem are extremely encouraging, demonstrating the ability for MTD techniques to solve problems with thousands of tasks.

Three key insights allowed us to approximately solve large MDPs in this fashion. The first is the ability to decompose the process into pseudo-independent subprocesses, and construct optimal policies and value functions for these subMDPs feasibly. Often special features of the domain (in this case, the noisy-or dynamics and limited windows) can be exploited to solve these subMDPs effectively. The second is that these value functions can offer guidance in the construction of policies that account for the interactions between processes. Again, special domain features (here, the convexity of the value functions) can offer guidance regarding the appropriate heuristic techniques. The third is the use of on-line policy construction to alleviate the need to reason about many fu-

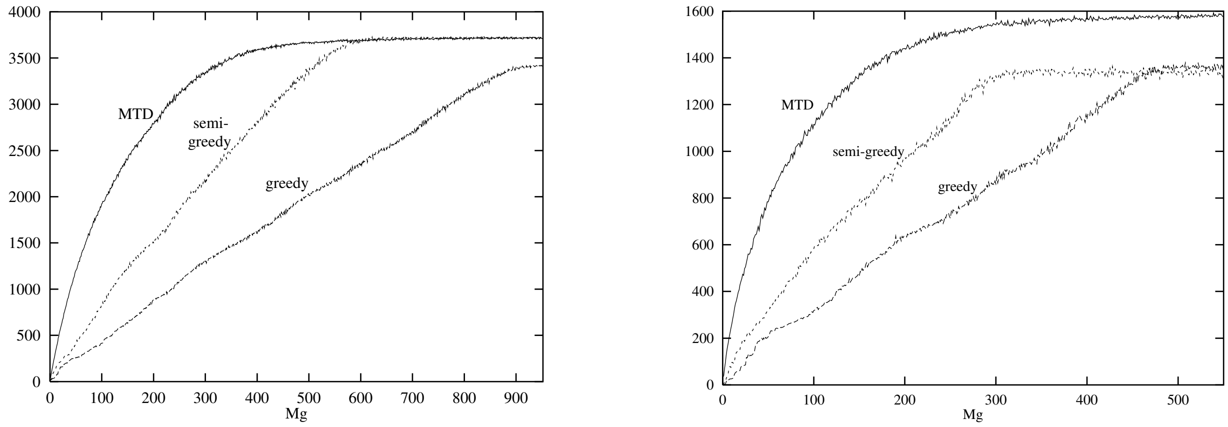

Figure 4: Comparison of the quality of policies generated by MTD, greedy and semi-greedy strategies on large test problems. The graph on left shows results for a 200-task problem with no local constraints. The graph on the right describes a 100-task problem with local constraints. Values are averaged over 100 runs.

ture contingencies. While on-line methods are popular [1], crucial to the success of the on-line component of MTD is the ability to quickly construct good actions heuristically using the component value functions.

MTD is a family of algorithms that exploit specific structure in the problem domain to make decisions effectively. It requires that the problem be specified in a specific form, taking advantage of utility independence and probabilistic independence in action effects. Much recent research has focussed on using representations for MDPs that make some of this structure explicit and automatically discovering appropriate problem abstractions and decompositions [9, 6, 14, 11, 5]. The extent to which effective Markov task decompositions can be automatically extracted from suitable problem representations remains an interesting open question.

References

[1] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[2] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[3] D. A. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London, 1985.

[4] D. P. Bertsekas and J.. N. Tsitsiklis. *Neuro-dynamic Programming*. Athena, Belmont, MA, 1996.

[5] C. Boutilier, R. I. Brafman, and C. Geib. Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. *Proc. IJCAI-97*, pp.1156–1163, Nagoya, 1997.

[6] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. *Proc. IJCAI-95*, pp.1104–1111, Montreal, 1995.

[7] T. Dean and R. Givan. Model minimization in markov decision processes. *Proc. AAAI-97*, pp.106–111, Providence, 1997.

[8] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76:35–74, 1995.

[9] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[10] T. Dean and S. Lin. Decomposition techniques for planning in stochastic domains. *Proc. IJCAI-95*, pp.1121–1127, Montreal, 1995.

[11] R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 89:219–283, 1997.

[12] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[13] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1978.

[14] A. E. Nicholson and L. P. Kaelbling. Toward approximate planning in very large stochastic domains. *AAAI Spring Symp. on Decision Theoretic Planning*, pp.190–196, Stanford, 1994.

[15] D. Precup and R. S. Sutton. Multi-time models for temporally abstract planning. In M. Mozer, M. Jordan and T. Petsche editor, *NIPS-11*. MIT Press, Cambridge, 1998.

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

[17] S. P. Singh and D. Cohn. How to dynamically merge markov decision processes. In M. Mozer, M. Jordan and T. Petsche editor, *NIPS-11*. MIT Press, Cambridge, 1998.