Toward Hierachical Decomposition for Planning in Uncertain Environments

Terran Lane and **Leslie Pack Kaelbling**
MIT Artificial Intelligence Laboratory
Cambridge, MA, 02139
USA
{terran,lpk}@ai.mit.edu

Abstract

This work addresses the computational complexity of solving large Markov decision processes by partitioning their state spaces into nearly-independent regions. We propose a method for developing partial plans, or macros, over such regions that yields a polynomial number of macros for each region. Each macro represents knowledge about how to achieve one particular sub-goal within its region of interest, and we demonstrate a simple, linear-time rule for combining these macros into a single full policy over the region and for propagating value information between regions. We empirically demonstrate the efficacy of our approach on small MDPs, for which exact solution is feasible, and give asymptotic analyses of its scalability.

1 Introduction

Markov decision processes (MDPs) have proven to be effective and principled tools for modeling many types of planning and action problems that are subject to uncertainty [Boutilier *et al.*, 1999]. It has become clear, though, that one of the largest barriers facing the widespread deployment of these models is a lack of scalability. While exact solution techniques for MDPs run in time and space that is polynomial in the number of model states, the number of states is often exponential in quantities of real interest such as the number of domain variables. Furthermore, often for explicitly representable and relatively modest problems, even the polynomial solution time may be unacceptable (as in realtime planning and execution).

Thus, there has recently been intense interest in methods for improving the scalability of MDPs. Researchers have proposed techniques based on factoring the transition model of the MDP [Koller and Parr, 2000], exploiting irrelevant variables [Boutilier and Dearden, 1994; Dietterich, 2000], or approximating the model's value function [Bertsekas and Tsitsiklis, 1996]. We draw on two other traditions, both based on developing *macros*, or local plans for accomplishing limited sub-goals over restricted, semi-autonomous regions of the state space. The first follows Kaelbling's HDG work [1993] which exploits a nearest-neighbor relation to partition the state space for approximately solving the point-to-point navigation task. HDG, however, does not handle propagation of information between regions and is restricted to unitary rewards. Moore et al. relax some of these restrictions with their airport data structure [1999], which encodes knowledge of how to (sub-optimally) navigate between any pair of states in the MDP. They are concerned with one-shot goals of achievement and do not handle tradeoffs between different goals, nor do they allow modification of the reward function. The second tradition follows the work of Precup et al. [Precup, 2000; Sutton *et al.*, 1999], which shows how to knit together a hand-crafted set of regions and macros into a single plan by solving a meta-level decision process. This approach correctly handles propagation of information between regions and both Hauskrecht et al. and Parr extend it with methods for automatically developing macros by selecting sub-goals as macro targets. The latter approaches require certain limiting assumptions and yield, typically, exponentially many separate macros for a given region. Our goal is to leverage the strengths of these two approaches: we use the navigational insights of HDG and the airport work, accepting their sub-optimal results, but incorporate the region and macro-handling approach of Precup et al. which allows us to treat more general goal functions. The approximations that lead to sub-optimal performance allow us to generate only polynomially many macros, rather than the exponential number that Hauskrecht et al. and Parr require to achieve near-optimality.

2 Separating Planning and Acting

We address the problem of planning in domains in which the world dynamic — the state and action spaces and transition effects of the actions — is fixed and reward function of the MDP is unknown, but the *states* in which reward can be received are limited to a small, fixed subset of the full state space. In other words, the world is stationary, but the goals or tasks of the agent change over time. Navigation domains, in particular, often demonstrate this property — an agent may exist in a limited, unchanging map with a fixed set of primitive actions, but be required to navigate to different locations at different times or with changing priority. For example, a delivery robot might know that there are a fixed set of possible destinations, but which are to receive packages (i.e., generate reward) depends on the particular assortment of packages on a given day. We are attempting to separate the navigation problem from the particulars of reward assignments to improve aggregate performance over multiple problem solving instances. We exploit two properties to achieve this separation: approximate independence of goals and a sparse, Markov transition function. The first allows us to develop separate navigation macros that describe how to seek each goal alone, while the second allows us to isolate the effects of rewards in one subset of the state space from other areas. Note that, though we conceptualize this approach in terms of robotic navigation, it is not restricted solely to such domains. Any fixed domain with multiple goals of achievement meets these conditions (though we introduce other restrictions later that may be more difficult to fulfill).

We refer to the set of goal states without reward assignments as a *problem domain* and the combination of goal states with specific reward values as a *problem instance*. Our model is a slight generalization of that employed by, e.g., Precup et al. [2000; 1999], Hauskrecht et al. [1998], or Parr [1998], who all develop macros only by considering the exit periphery states (e.g., "doors") between macro-regions, but who do not account for goal states that may occur within a macro-region. We also put different constraints on possible reward values than do some of these prior authors: Hauskrecht et al. and Parr both constrain the possible values occurring at a peripheral state to fall within some bounded range while we bound possible values only to be non-negative.

Intuitively, our approach depends on the observation that, for a problem domain with a *single* goal state, the structure of the optimal policy for achieving that goal is independent of reward *scale*[1]: the same series of actions maximize aggregate reward whether the particular reward instance for that goal is one or one thousand. A macro for achieving a particular goal remains suitable no matter how the reward value is scaled. We therefore develop one macro for achieving each potential sub-goal within the full MDP *independently* from all other sub-goals. In a partitioned MDP state space, we also develop independent macros for moving from every region to each of its neighbors. Given a particular instantiation of a real reward function (e.g., a particular assortment of packages and places to deliver them), we re-scale the individual macros in proportion to their instantiated rewards and combine them into a full policy in time and space that is linear in the number of states.

Of course, sub-goals are *not* truly independent and ignoring one while achieving another may be deleterious. To fully address this difficulty requires handling the combinatorics of multiple interacting sub-goals. The approaches of Hauskrecht et al. and Parr treat these interactions exactly and yield macro caches that achieve an ϵ-optimal value for any possible reward function (over a bounded range), but they cannot escape the combinatoric difficulty and, in the worst case, their approaches produce macro caches that are exponentially large in the number of exit periphery states. We explicitly sacrifice optimality in favor of a guarantee of a polynomial number of macros. Parr does demonstrate how to begin with a few macros for a region and to add new macros when the current cache is found to be inadequate for a particular problem instance. Unfortunately, his approach requires solving a linear program to detect inadequacy and one or more MDP planning problems to update the cache — both relatively expensive operations — and puts no bound on the number of macros that the cache will accumulate over time. Our approach retains a small fixed set of "basis" macros from which we compose a final plan for a region in a single, linear time pass.

[1]Though it does depend on the sign of the reward: negative rewards, or penalties, yield structurally different policies than do positive rewards. Currently, we address only non-negative rewards.

3 Formal Model

Here, we overview the terminology we use in this paper. We briefly summarize the background of Markov decision process theory; for a more in-depth discussion of MDPs, we refer the interested reader to Puterman's text [1994].

3.1 Markov Decision Processes

A Markov decision process, $\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$, is a description of a synchronous control domain specified by four components: a *state space*, $\mathcal{S} = \{s_1, s_2, \dots, s_N\}$, of cardinality $|\mathcal{S}| = N$; a set of primitive *actions*, $\mathcal{A} = \{a_1, a_2, \dots, a_k\}$, of cardinality $|\mathcal{A}| = k$; a *transition function*, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$; and a *reward function*, $R : \mathcal{S} \to \mathbb{R}$. The transition function, written $T(s'|s, a)$, determines the probability of arriving in state s' upon taking action a from state s and must represent a valid probability distribution: $\forall s, a \ \sum_{s' \in \mathcal{S}} T(s'|s, a) = 1$. An agent acting in a given MDP is, at any time step, located at a single state $s \in \mathcal{S}$. The agent chooses an action $a \in \mathcal{A}$ and is relocated to a new state, s', determined by the transition probability distribution $T(s'|s, a)$, whereupon it receives reward $R(s')$. The goal of the agent is to maximize its aggregated reward over time. There are a number of models of time-aggregated reward, but we will address only *infinite horizon discounted* rewards in which the agent's expected future reward is discounted by a constant multiplicative factor, $0 \le \gamma < 1$, at each time step. The total reward received over an infinite operational lifetime is $r_{\text{tot}} = \sum_{t=0}^{\infty} \gamma^t R(s_t)$, where s_t is the state the agent reaches at time t. The goal of planning in an MDP context is to locate a *policy*, $\pi : \mathcal{S} \to \mathcal{A}$, that determines an action for the agent for any possible state. In general, π can specify a distribution over actions but an important theorem of MDP planning says that, for a fixed MDP, $\mathcal{P}$, there is a policy, $\pi^*_{\mathcal{P}}$, that maximizes $E[r_{\text{tot}}]$ and that that policy is *deterministic* [Puterman, 1994]. Associated with a given MDP and policy is a *value function*, $V^\pi : \mathcal{S} \to \mathbb{R}$, which records the total expected aggregate reward achieved by an agent that starts in state s and acts according to π forever. The value function for a particular MDP and policy can be found exactly by solving the *Bellman equations*, requiring $O(N^3)$ time and $O(N^2)$ space.

In this work, we are concerned with MDPs having only partially specified reward functions. In particular, we assume that we are given a set of *goal states*, $\mathcal{G} = \{g_1, g_2, \dots, g_\nu\}$, but that many possible reward functions are allowed, subject only to the constraints that $\forall s \in \mathcal{S}, R(s) \ge 0$ and $\forall s \in (\mathcal{S} \setminus \mathcal{G}), R(s) = 0$. We will refer to an MDP with only a specification of a set of goal states, $\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, T, \mathcal{G} \rangle$ as a *problem domain* and an MDP with a fully specified reward function (under the above constraints), $\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$, as a *problem instance*.

For the purposes of this discussion, we will refer to the original MDP as the *base-level* or *primitive* MDP and its associated states and actions as *atomic* states and actions. Regions and macros, which we define fully below, constitute meta-states and meta-actions, respectively, in a higher-level semi-Markov decision process.

3.2 Sub-Goals, Macros, and Macro Caches

In our approach, a macro is a plan for achieving one goal state in isolation. To model this, we define *sub-goal* as a restriction of the reward function of an MDP to a single state, g: $R_g(s) = 1$ when $s = g$ and $R_g(s) = 0$ otherwise. (As noted above, for a *lone* goal state, the scale of $R(g)$ is immaterial to the optimal plan for achieving it.) A *macro*, π_g, for achieving a sub-goal, g, over an MDP, $\mathcal{P}$, is a policy over $\mathcal{P}$ where R is restricted to R_g, and can be found via a standard technique such as policy iteration. A *macro cache*, $\mathcal{C} = \{\pi_{g_1}, \pi_{g_2}, \dots \pi_{g_\nu}\}$ is simply a set of such macros for achieving distinct goals $g_1, g_2, \dots g_\nu$.

3.3 Regions, Peripheries, and Partitions

By *region* we will denote a subset of the state space, $\mathcal{P} \subseteq \mathcal{S}$, along with the corresponding transition and reward functions when restricted to $\mathcal{P}$, $T_\mathcal{P} : \mathcal{P} \times \mathcal{A} \times \mathcal{P} \to [0,1]$ and $R_\mathcal{P} : \mathcal{P} \to \mathbb{R}$. (When necessary for clarity, we will use subscripts to denote the underlying state space of a function, e.g., $T_\mathcal{S}$ vs. $T_\mathcal{P}$.) We add a special, zero-reward absorbing state, σ, to each region and direct all outgoing transitions to it: $\forall s \in \mathcal{P}, T_\mathcal{P}(\sigma|s,a) = \sum_{s' \in \mathcal{S} \setminus \mathcal{P}} T_\mathcal{S}(s'|s,a)$. The reward function is the corresponding restriction of the base-level reward function: $\forall s \in \mathcal{P}, R_\mathcal{P}(s) = R_\mathcal{S}(s)$. Thus, the region $\mathcal{P}$ itself constitutes a proper Markov decision process and we can define policies and macros over it. We denote the set of goal states occurring within a region by $\mathcal{G}_\mathcal{P} = \mathcal{G} \cap \mathcal{P}$.

The *exit periphery* of a region, $X_\mathcal{P}$, is the set of states having non-zero transition probability out of a region: $X_\mathcal{P} = \{s \in \mathcal{P} : T_\mathcal{S}(s'|s,a) > 0\}$ for some $s' \in \mathcal{S} \setminus \mathcal{P}$. Informally, the exit periphery states are "doorways": states that must be reached before an agent can transition from one region to another. Corresponding to exit peripheries are entrance peripheries: $I_\mathcal{P} = \{s \in \mathcal{P} : T_\mathcal{S}(s|s',a) > 0\}$ for some $s' \in \mathcal{S} \setminus \mathcal{P}$. The periphery states are a separating set: the value function within a region is independent of the value function outside the region, given the values at its periphery states.[2]

A *partition* of an MDP is a division of the MDP into separate regions that collectively cover the state space: $\mathcal{S} = \bigcup_i \mathcal{P}_i$. We constrain the regions to be disjoint except for their periphery states: $\forall i \neq j$, $\mathcal{P}_i \cap \mathcal{P}_j \subseteq X_{\mathcal{P}_i} \cup X_{\mathcal{P}_j}$, and we note that every exit periphery state of some region corresponds to an entrance periphery state of another region.

[2]In the sense that, given the value function at the exit periphery states for a region, one can exactly calculate the value function at all other states within that region with no further information. In particular, given the exit state values, knowledge of value functions outside the region is irrelevant to computation of the value function within the region.

4 Applying Macros and Regions

We can now describe our planning and execution algorithm in full detail. Local macro planning handles trading off the importance of achieving different goals within a single region, while the region-integration algorithm integrates the local results into a global plan.

4.1 Building and Executing from a Macro Cache

Our model of macro planning differs slightly from that of Precup et al., Hauskrecht et al., and Parr. They treat macros as discrete actions that "solve" a given MDP or region in a particular way and select a single macro per region for each problem instance. Our approach treats macros as building-blocks which, given a problem instance, we merge to form a single policy for the region. In an intuitive sense, you can think of our macros as being "basis policies" from which we construct full policies for a region.

Given an MDP problem domain $\mathcal{P}$ with goal states $\mathcal{G} = \{g_1, g_2, \dots, g_\nu\}$, we construct a macro cache corresponding to the goal states of $\mathcal{P}$: $\mathcal{C}_\mathcal{G} = \{\pi_{g_1}, \pi_{g_2}, \dots, \pi_{g_\nu}\}$. Intuitively, each of these macros represents the "shortest path" information from each atomic state to one particular sub-goal. Associated with each macro is its corresponding value function, V^g.

Now, given a problem instance, we re-scale each macro's value function according to the instantiated reward of its goal state: $\widetilde{V}^g(s) = R(g)V^g(s)$ for all $s \in \mathcal{S}$. In general, we will use a tilde to indicate a function after incorporating the effects of a problem instance, while the plain name indicates the instance-independent version. $\widetilde{V}^g$ is now a measure of how "close" every state is to g relative to the importance of attaining g under R. We wish to construct a single policy, $\widetilde{\pi}$, that combines the information from each macro. The simplest rule, which we employ here, is to choose the action corresponding to the macro with the greatest value at each state: $L(s) = \arg\max_{g \in \mathcal{G}}\{\widetilde{V}^g(s)\}$ and $\widetilde{\pi}(s) = \pi_{L(s)}(s)$. In a deterministic, continuous control process, this corresponds to splitting the world into a Voronoi partition around the goal points and greedily choosing actions to seek the nearest goal. Unfortunately, MDPs are neither deterministic nor continuous so this rule is only an approximation and it can yield substantially suboptimal policies. Nevertheless, we find that it performs reasonably well in empirical investigations, as we demonstrate in Section 5. This rule does have the benefit of being quite efficient to implement; for each problem instance it requires time and space linear in the number of atomic states, while exact solution requires greater than cubic time and quadratic space. In fact, we need not even fully generate $\widetilde{\pi}$ — we can simply compute the appropriate action for the agent to take at its current position given our macros and our knowledge of the reward instantiation. This is still not good enough to overcome the exponential size of many interesting state spaces, however, so we turn our attention to partitions, which offer the possibility of hierarchically decomposing an MDP and realizing exponential improvements.

```
**Procedure** *RegionPlan*($\mathcal{P}$)
**if** $\mathcal{P}$ is primitive
    **foreach** $e \in I_{\mathcal{P}}$
        $\widetilde{R}_{\mathcal{P}}(e) = \max_{g \in \mathcal{G}_{\mathcal{P}}} \{\widetilde{V}^g(e)\}$
**else**
    **foreach** sub-region, $\mathcal{P}_i$ of $\mathcal{P}$
        *RegionPlan*($\mathcal{P}_i$)
    *SolveShortestPaths*($\mathcal{G}_{\mathcal{P}} \bigcup_i X_{\mathcal{P}_i}$,$\widetilde{V}$)
```

Figure 1: Pseudocode for the region planning algorithm.

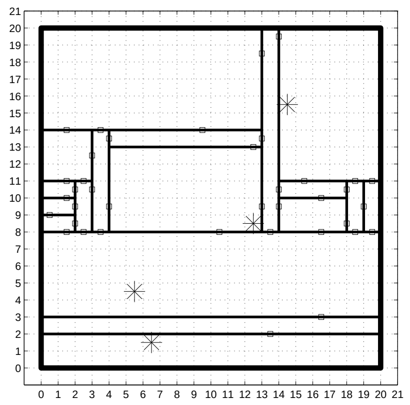

Figure 2: An example synthetic maze. The star symbols denote goal states, lines and thick lines are walls, and small squares are “doors” which allow transitions between rooms in spite of walls.

4.2 Planning and Acting with Regions

We first note that individual regions, as we have defined them, constitute complete MDPs, so we can construct macro caches over them, as described above. The question becomes one of communication between regions: how does the reward function instantiation in one region affect the choice of macros in other regions? For this purpose, we follow Hauskrecht et al. and use the exit periphery states. In their model, macros are treated as discrete meta-actions and regions as meta-states which together constitute a semi-Markov decision process. As Precup et al. have demonstrated, solution of the semi-MDP selects a specific macro to execute in each region and yields a plan that is optimal within the space of policies representable with only the available macros.

In our algorithm, macros are not thought of as discrete actions but as parts to be combined into a single policy specific to a region and particular reward instantiation. Planning in the partitioned space is not a question of choosing between macros, but of propagating the influence of the reward function between regions. Fortunately, this influence is quantified for us by the value function, V^g, associated with goal g. We can use the periphery states, $I_{\mathcal{P}}$ and $X_{\mathcal{P}}$, to propagate this value into adjacent regions. Recall that exit periphery states exist in multiple regions simultaneously, and that an exit periphery state for some region $\mathcal{P}_j$ is also an entrance periphery state for some other region $\mathcal{P}_i$. The value function induced by a goal state, $g \in \mathcal{P}_i$, at an entrance periphery state, $s \in I_{\mathcal{P}_i}$, can thus be treated as a reward at its corresponding exit state, $s = x \in X_{\mathcal{P}_j}$. We treat exit periphery states as “synthetic goals”: placeholders that have no real reward function themselves, but to which we assign estimated reward based on the real rewards of a problem instance. We denote the synthetic reward imputed at exit state x by $\widetilde{R}_{\mathcal{P}_j}(x)$. It is important to note that this synthetic reward exists *only* in the region for which x is an exit periphery state, not in any regions for which x is an entrance periphery state. Thus, an agent has incentive to leave a region via an exit state, but no incentive to return once it leaves.

The combination of goals and synthetic goals, coupled with the value functions, forms a weighted graph. The insight of HDG and the airport data structure is that we can treat such graphs as deterministic distance graphs (V is inversely related to distance via γ) and solve them efficiently with shortest-path algorithms. This is approximate because of non-determinism, but for large distances and fairly uniform transition functions, the law of large numbers ensures achieved values will be close to our estimates. This leads us to a natural, recursive algorithm for planning with regions, given in Figure 1. Regions are either primitive (possessing no sub-partitions) or complex. For a primitive region, $\mathcal{P}$, we apply the value-estimation rule of Section 4.1 to determine the influence of goals within R on its entrance periphery. For a complex region, we recursively plan over its sub-regions and then apply a modified shortest-path solver to determine which goal is most important to achieve from each periphery state, x. These estimates now form $\widetilde{R}(x)$, which allows us to trade off heading for local goals against heading for exits when acting.

We note that the planning algorithm does not form a full policy over the MDP, but only finds value estimates at exit periphery states. The agent acts locally by choosing between macros associated with the goal and periphery states local to its current primitive region, as in Section 4.1. Thus, action is nearly constant time and planning relative to a given problem instance is polynomial in the number of goal and periphery states per region. We discuss the asymptotics of this algorithm in Section 6.

5 Empirical Performance Evaluation

Although we are ultimately interested in handling extremely large MDPs, we begin by empirically examining the performance of our planning mechanisms in small domains for which exact solution is still feasible. We examine only the quality of solution here; we defer issues related to computational complexity and asymptotic time and space performance to Section 6. These results are preliminary — they examine only small, geographic MDPs and only binary partitions of the state space — but they provide a proof-of-concept demonstration that our planning heuristics operate reasonably in some plausible spaces. In Sections 6 and 7 we discuss the scalability of these results and the types of MDPs we expect to be amenable to our approach.

We have examined the performance of our approach on a number of synthetic maze domains of the type illustrated in Figure 2. This is a grid-world maze with four actions available from each state: North, South, East, and West. If the selected action succeeds (with probability p), the agent is taken to the state in the indicated direction; if it fails, the agent is deposited in one of the three other orthogonal states with uniform probability. Walls are impermeable and an attempt to move in the direction of a wall (deliberately or accidentally) leaves the agent's state unchanged. Doors are simply holes in the wall — they allow unimpeded progress in both directions.

We generated mazes with between 25 and 900 states, p ranging between 0.5 and 0.9, and a variable number of rooms and corridors.[3] In this framework, we performed two experiments: one examining the performance of our macro planning system on an unpartitioned state space and one which splits the state space into two regions and builds macros separately for each region.

In the first experiment, we randomly selected 2, 3, or 5 goal states in the maze and assigned them random rewards in the range $(0, 100]$. We solved for the optimal policy, π^*, with standard policy iteration and built a macro cache as described in Section 4.1. We constructed a full policy for the MDP, $\pi^{\mathcal{C}}$, by combining the macros using the assigned rewards for their sub-goals, and evaluated that policy by solving the Bellman equations for the value function at each state, $V^{\mathcal{C}}(s)$. We summarize the performance of a particular policy with *regret*:

$$\left| \frac{\max_{s \in \mathcal{S}} \{V^*(s) - V^{\mathcal{C}}(s)\}(1-\gamma)}{\max_{g \in \mathcal{G}} \{R(g)\}} \right| .$$

This quantity expresses the difference between the optimal and constructed policies in terms of the *Bellman error* (the maximum difference in value functions) scaled in proportion to the maximum reward achievable in the MDP. The factor of $1-\gamma$ scales value functions into the same range as reward functions.

Over the 864 mazes we generated in this experiment, the mean regret was only 1.4%, indicating that our policy construction technique is performing reasonably well over a range of conditions in these small mazes. The largest variation in regret appears to be related to the distance between the goal states; we found a roughly inverse relation between regret and the minimum goal distance (measured in Manhattan distance). This is a consequence of the greedy goal selection we use: for distant, well separated goals, the optimal strategy is to head for one and stay there when you reach it. Optimally handling close goals, however, requires subtler plans than our heuristics can generate.

In the second experiment, we focussed on the largest of the mazes that we could solve optimally, generating mazes with 400, 600, or 900 states, and introduced only two goal states. We partitioned the maze into two regions by locating the smallest set of doors that separated the two goal states using a max-flow/min-cut algorithm [Cormen *et al.*, 1992]. The states on either side of these doors formed the peripheries of the two regions. Now, with goal and periphery states for each region in hand, we developed a macro cache for each region as discussed in Section 4.2 and evaluated it in terms of regret with respect to an optimal policy for the full maze. In the 360 MDPs we generated for this experiment, the mean regret of the macros-over-partitions policies was 0.4% — larger than the regret of macros over an unpartitioned state space (which achieved 0.03% regret on these same mazes), but not catastrophic.[4] We again observed an inverse relation between regret and Manhattan distance, though now doors (i.e., exit periphery states) must also be considered goals.

6 Toward Very Large State Spaces

The empirical results we have presented demonstrate that our macro planning system performs reasonably, albeit suboptimally, in small state spaces. This represents but a first step toward our ultimate goal of treating extremely large state spaces; in our ongoing work we are extending our implementation to handle very large domains (e.g., domains specified by a large number of discrete variables whose cross-product yields an exponentially large state space). In this section, we give asymptotic arguments about how we expect our techniques to perform in these domains.

The major leverages that we intend to exploit in scaling these algorithms are those provided by hierarchical partitioning and solution reuse. Single-shot exact solution of an MDP with N states requires at least $O(N^3)$ time and $O(N^2)$ space (to solve for the policy and store it) while our macro planning method requires $O(\nu N^3)$ time and $O(N^2 + \nu N)$ space for an unpartitioned MDP with ν goal states. The important improvement comes when we consider regions: assume that a region (e.g., an entire MDP) with ν goal states is partitioned into ρ regions, each of size $O(N/\rho)$, and that there are a total of X periphery states connecting the regions. If we solve the shortest path problem with ν applications of Dijkstra's algorithm [Cormen *et al.*, 1992], the time to execute the planning algorithm for a given reward instance is recursively expressed as $T(N) = \rho T(N/\rho) + \nu X \log(X)$. If the regions are sparsely connected and $X = O(\rho)$, then this recurrence has the solution $T(N) = \Theta(N)$. Thus, we spend only linear, rather than cubic, time per problem instance. Pre-processing a problem domain to locate the macros initially is also efficient: if partitioning stops when the MDP has been split into $O(N/c)$ regions of some constant size, c, then the time to generate all macros is $O(Nc^3)$ and they can be stored in space $O(Nc^2)$. Furthermore, this step is easily parallelizable: the solution of one macro is independent of all others. Finally, we can probably achieve yet more improvements by ignoring parts of the hierarchy irrelevant to a particular problem instance or by caching the value function only at periphery states and regenerating macros locally as needed.

These results represent a substantial improvement in the complexity of the solution of MDPs. When handling mutable reward functions with solution reuse, we achieve one-shot linear planning time and at least linear per-episode execution

[3]900 states is the limit of our, admittedly non-optimized, code's ability to exactly solve the MDP for an optimal policy.

[4]We attribute the improvement in regret from the previous experiment to larger mazes which lead to a greater proportion of large inter-goal distances and small regrets.

time, as opposed to cubic per-episode planning time. A critical question is how to hierarchically select regions. For best performance, we must have roughly balanced regions with small peripheries. This question has not been well addressed in the literature because it is hard to define what constitutes a "good" partition. We believe that our asymptotic results can provide an operational definition — an optimization criterion based on estimated solution time and space complexity.

Unfortunately, our best asymptotic results are still insufficient for extremely large state spaces where N itself is exponential in the number of state variables and perhaps even enumerating the state space is impossible. This case is difficult for all of the region-based macro methods we have discussed because they all need to examine every state at least once when constructing macros. We must look for further extensions for progress in this regime. Possibilities include solving for macros only over a very limited subset of the available regions; discarding some periphery states; reusing macros between structurally similar regions; approximating the value function over some regions with regression functions, as in neuro-dynamic programming [Bertsekas and Tsitsiklis, 1996]; or searching only over relevant subsets of the state variables, as do Dietterich [2000] and Boutilier and Dearden [1994].

7 Conclusions and Future Directions

We have demonstrated a method for decomposing large state spaces in two ways: spatially, in which the state space is broken up into semi-autonomous regions, and teleologically, in which macros are independently developed to solve each subgoal of the MDP. The first offers the possibility of decomposing the state space into tractable fragments which can be addressed independently and can be quickly reintegrated during an action phase. The second offers the ability to quickly adapt to changing reward functions (e.g., in repeated problem solving instances), given that the locations of the goal states are fixed. By employing both, we can potentially realize a polynomial reduction in the complexity of planning, which is now a one-time operation per problem domain, can incorporate new reward functions in linear time, and can choose actions in nearly constant time. We have given proof-of-concept empirical results which demonstrate that these heuristics incur only slight penalties, at least in small maze domains.

Clearly, the question of real scalability of these methods is yet to be answered. We foresee no difficulty, in principle, in scaling them up to larger maze worlds. In general, we expect these heuristics to work well in MDPs with the following characteristics:

Non-negative rewards The macro-action rule we describe in Section 4.1 turns out to be suitable only for non-negative rewards. Negative rewards (penalties) are troublesome because a purely greedy approach — head directly away from the penalty — is not typically effective, in the context of other sub-goals.

Sparse reward function It is critical to the scaling of our approach that the MDP have only a very few goal states, as we must introduce a macro for each. Furthermore, as we demonstrated empirically, the goal states should be well separated (in terms of expected number of steps to transition between different goals), so that there is little chance of deleterious interactions among them.

Many regions with sparse peripheries The greatest benefit will be achieved when the maze can be partitioned into many small regions which can be solved independently. It is important, though, that the regions have sparse peripheries — i.e., there be few transitions between different regions relative to the size of each.

Metric transition functions Our planning heuristics effectively use value function as a distance. In general, an MDP's transition function need not represent a metric space — it can have irreversible actions and "teleporters" that subvert transitivity. How well our approach will apply in such spaces is still an open question.

There are many useful domains that exhibit these qualities; any task based on spatial navigation, especially in buildings or street maps, with limited goals of achievement, would be suitable, for example. We are currently working on simulators for such domains with much larger state spaces, which will provide a better evaluation of these methods as well as a testbed for examining other techniques. In addition, we believe that we can relax some of the aforementioned restrictions. In particular, we are examining approaches which may allow us to handle negative rewards and constant rewards over large regions of the state space (e.g., a constant movement cost due to battery drain).

References

[Bertsekas and Tsitsiklis, 1996] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Optimization and neural computation series. Athena Scientific, Belmont, MA, 1996.

[Boutilier and Dearden, 1994] C. Boutilier and R. Dearden. Using abstractions for decision-theoretic planning with time constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1016–1022. AAAI Press, 1994.

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[Cormen *et al.*, 1992] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1992.

[Dietterich, 2000] T. G. Dietterich. An overview of MAXQ hierarchical reinforcement learning. In B. Y. Choueiry and T. Walsh, editors, *Proceedings of the Symposium on Abstraction, Reformulation and Approximation SARA 2000*, Lecture Notes in Artificial Intelligence. Springer Verlag, New York, 2000.

[Hauskrecht *et al.*, 1998] M. Hauskrecht, N. Meuleau, C. Boutilier, L. P. Kaelbling, and T. Dean. Hierarchical solution of Markov decision processes using macro-actions. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty* *in Artificial Intelligence (UAI-98)*. Morgan Kaufmann, 1998.

[Kaelbling, 1993] L. P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173, 1993.

[Koller and Parr, 2000] D. Koller and R. Parr. Policy iteration for factored MDPs. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference (UAI 2000)*. Morgan Kaufmann, 2000.

[Moore *et al.*, 1999] A. W. Moore, L. C. Baird, and L. Kaelbling. Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden, 1999. Morgan Kaufmann.

[Parr, 1998] R. Parr. Flexible decomposition algorithms for weakly coupled Markov decision problems. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*. Morgan Kaufmann, 1998.

[Precup, 2000] D. Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, Department of Computer Science, 2000.

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

[Sutton *et al.*, 1999] 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.