Research Abstract: Hierarchical Solution of Large Markov Decision Processes

Jennifer Barry
Keywords: Markov Decision Processes, Sequential Decision Making, Planning Under Uncertainty

1 Introduction

Our goal is to solve a large class of very large Markov decision processes (MDPs), necessarily sacrificing optimality for feasibility. It is widely believed that hierarchical decomposition is the key to solving very large planning problems. Several approaches have shown that, given an appropriate hierarchical decomposition in advance, it can indeed provide considerable speed-up in solving an MDP (Parr 1998; Hauskrecht et al. 1998; Lane and Kaelbling 2002) or doing reinforcement learning (Parr and Russell 1997; Sutton, Precup, and Singh 1999; Dietterich 1998). Finding an appropriate hierarchy has proved challenging, however. There are recent approaches to learning hierarchies (McGovern and Barto 2001; Simsek, Wolfe, and Barto 2005; Mehta et al. 2008); they operate over a relatively long timescale as the agent has to learn the world dynamics in the process, making this problem substantially more difficult. The expectation is that the work associated with learning the hierarchy will pay off over the course of solving several related problems in the same or similar domains.

Our goal in this work is somewhat different. We would like to solve a single large MDP very quickly. As the size of the MDP increases, exact solution becomes intractable, so we expect only to find an approximate solution. If we are to use hierarchy to solve this problem, we must be able to *construct and solve* the hierarchical model in less time than it would have taken to simply solve the original flat model.

Bakker et al. (Bakker, Zivkovic, and Krose 2005) presented an approach to this problem, which works well on two-dimensional navigation problems but does not seem to generalize well to other types of domains, sometimes failing to find any strategy, even a suboptimal one, for an achievable goal. Jonsson and Barto's VISA algorithm (Jonsson and Barto 2006) addresses the problem by taking as input a factored MDP model described as a DBN, and doing causal analysis to find a good hierarchical decomposition. The work of Maggioni and Mahadevan (Mahadevan 2008) finds a multi-scale basis for representing value functions in MDPs and other problems; this basis can serve as an effective representation for solving the MDP efficiently, but is not hierarchical in the same sense as the other methods discussed here.

Typical approaches to solving hierarchical MDPs work bottom-up. But the problem is that because it is not yet known what high-level actions will be selected, the low-level problems must be solved many times with many different possible high-level objectives. We take advantage of a quick bottom-up pass based on a deterministic approximation of expected costs to move from one state to another, and use that to derive a policy from the top down, which avoids solving low-level MDPs for multiple objectives. The resulting policy may be suboptimal but it is guaranteed to reach a goal state in any problem in which it is reachable under the optimal policy.

We begin by describing our conception of a hierarchical model of an MDP and how we can create and solve this model for enumerated-states MDPs. We will then briefly discuss our current work on factored MDPs.

2 Hierarchical Model

A Markov decision process (MDP) is defined by $\langle S, A, T, R\rangle$, where S is a finite set of states, A is a finite set of actions, T is the transition model with $T(i', a, j')$ specifying the probability of a transition to i' given that the system starts in state j' and selects action a, and R is the reward model with $R(i', a)$ specifying the real-valued reward of taking action a in state i'. In addition, we assume a prespecified set of *goal states*, $G \subset S$. Goal states are zero-reward absorbing states: for every $g \in G$, $T(g, a, g) = 1$ and $R(g, a) = 0$, for all a. Further, we assume that all other reward values are strictly negative. We solve this problem under the undiscounted total reward criterion, making it a 'stochastic shortest path' problem. Any MDP can be transformed into an 'equivalent' stochastic shortest path problem, which can then be solved to produce the optimal policy for the original MDP (Bertsekas and Tsitsiklis 1996).

From the input MDP, we construct and then solve a *hierarchically determinized MDP* (HDMDP). An HDMDP with L levels is given by a depth-L *tree*. The leaves of the tree, at level 0, are the states of the original MDP, referred to as *primitive states*. Internal nodes of the tree represent (possibly overlapping) sets of nodes at the lower levels. We refer to nodes of the HDMDP as macro-states. The set of macro-states at level l is represented by S^l.

The solution process computes a *hierarchical policy* π with L levels, each of which prescribes behavior for each level l state. At levels $l > 0$, the policy π^l maps each level l macro-state i to some other level l macro-state j, signify-

Algorithm 1

Input: S^{l-1}: level $l-1$ states, A: primitive actions, T: transition function, G: primitive goal states

Output: A g-connected clustering of level l macro-states

ESCLUSTER(S^{l-1}, A, T, G)

1 $S^l \leftarrow \big\{\{i'\} \mid i' \in S^{l-1}\big\}$
2 *// create "goal macro-state" for level* 1
if $l = 1$, $g \leftarrow \{i' \mid i' \in G\}$, $S^1 \leftarrow \big(S^1 \setminus \{\{i'\} \mid i' \in G\}\big) \cup g$
3 **else** $g \leftarrow \{l-1$ goal macro-state$\}$ *// goal state already exists*
4 $Adj^l \leftarrow$ ADJMATRIX(S^l, A, T) *// adjacency defn in Sec. 2*
5 set g adjacent to every state
6 **while** $|S^l| >$ MINCLUST$_{ES}$ and $S^l_{\max} <$ MAXSIZE$_{ES}$
7 $\{y_1, y_2, ..., y_n\} \leftarrow$ FINDCYCLE(S^l, Adj^l)
8 $Y \leftarrow \{y_1, ..., y_n\} \setminus g$
9 create new macro-state $u \leftarrow Y$
10 $S^l \leftarrow \big(S^l \setminus Y\big) \cup u$, remove Y from Adj^l and add u
11 **if** $g \notin \{y_1, ..., y_n\}$
12 **for** i adjacent to some $y \in Y$, set i adjacent to u
13 **else** set only g adjacent to u
14 **for** i s.t. $\exists y \in Y$ adjacent to i, set u adjacent to i
15 **return** S^l

ing that when the system is in a primitive state contained in macro-state i it should attempt to move to some primitive state in macro-state j. At level 0, the policy π^0 is a standard MDP policy mapping the primitive states to primitive actions.

At the primitive level, a state i' is *adjacent* to a state j' if there is some action a such that $T(j', a, i') > 0$. At levels $l > 0$, a macro-state i is adjacent to a macro-state j if there is some $i' \in i$ and $j' \in j$ such that i' is adjacent to j'. A state j is *reachable* from a state i if j is adjacent to i or j is adjacent to some state k which is reachable from i. If $i' \in i$ is a level $l-1$ sub-state of i then a level $l-1$ state j' is reachable from i' if j' is adjacent to some state $k' \in i$ and k' is reachable from i' without leaving i.

3 Enumerated-States MDPs

3.1 Clustering Algorithm

We begin by discussing how we create and solve the hierarchical model for an enumerated-states MDP. We view creating the hierarchical model as clustering: macro-states at level l of the tree are clusters of level $l-1$ states. There are many plausible criteria for clustering states of an MDP, but we base our algorithm on one tenet: we want a structure in which every state that could reach a goal state in the flat MDP can reach a goal state under some hierarchical policy.

This criterion is *not* guaranteed by an arbitrary hierarchy and the type of hierarchical policy described in Section 2. That policy requires all sub-states of macro-state i at level l to find a path through i to some sub-state of $\pi^l(i)$. In a hierarchy where there is no level l state reachable from all sub-states of i, there is no hierarchical policy under which every sub-state of i can reach a goal state. To avoid such hierarchies, we require that, at each level, all macro-states be *g-connected*. A set U of macro-states with goal macro-state g is g-connected if there exists a policy $\pi : U \rightarrow U$ such that: (1) for all $i \in U$, i can reach g under π, and (2) for all $i \in U$, for each $i' \in i$ that can reach a goal state in the flat MDP, there exists $j' \in \pi(i)$ such that j' is reachable from i'.

Theorem 1: Let $S^0, ..., S^{L-1}$ be a hierarchical model for M such that the macro-states at each level are g-connected. Then there exists a hierarchical policy π such that for each $i' \in S$ that can reach a goal state in the flat MDP, i' can reach a goal state under π.

Proof: For a proof of this and all other theorems see (Barry, Kaelbling, and Lozano-Pérez 2010).

To create g-connected macro-states at level l from a set of $l-1$ macro-states, we run ESCLUSTER shown in Algorithm 1, which creates macro-states consisting of cycles of level $l-1$ states after setting the level l goal macro-state adjacent to all other level l macro-states. Setting the goal macro-state adjacent to all other states allows domains that contain few cycles to still be viewed hierarchically by grouping sets of states that are close together and lead to the goal.

Theorem 2: ESCLUSTER creates a g-connected clustering.

Proof Sketch: Each level l macro-state i is composed of a cycle of level $l-1$ states. If this is a true cycle, then all sub-states of i can reach all other sub-states in i and therefore a sub-state in any level l macro-state adjacent to i. Thus, in this case, all sub-states of i can comply with any policy that maps i to an adjacent macro-state. If i is composed of a "cycle" that goes through the goal macro-state g, all sub-states of i will be able to reach g. In this case, all sub-states of i will be able to comply with a policy that maps i to g. There is one subtlety: if i is composed of a cycle that goes through g, all sub-states of i can reach g, but may not be able to reach all macro-states adjacent to i. We acknowledge this in line 13 by marking only g as adjacent to i.

The complexity of ESCLUSTER is dominated by finding cycles, which is worst-case quadratic, but can be linear in domains where many states can reach a goal state.

Theorem 3: If a fraction p of the states in the MDP can reach a goal state, ESCLUSTER terminates in time $O(p|S| + (1-p)p|S|^2)$ where $|S|$ is the size of the state space.

ESCLUSTER relies on two user-defined parameters, MINCLUST$_{ES}$ and MAXSIZE$_{ES}$, defining the minimum number of macro-states allowed and the maximum size of those macro-states respectively. These parameters can be set to control the time/accuracy trade-off of the algorithm, as we will discuss in Section 3.3.

3.2 Solver

The hierarchical model created in ESCLUSTER is input for a solver that uses the g-connectedness to quickly find an approximate solution for the MDP. In solving, we approximate the cost of transitioning between upper-level macro-states as deterministic. This allows us to find policies for $l > 0$ quickly using a deterministic shortest path algorithm.

We run the algorithm in two passes as shown in Algorithm 2. In UPWARDPASS, we compute an approximation

Algorithm 2

Input: $S^0, ..., S^{L-1}$: hierarchical model, A: primitive actions, T: transition function, R: reward function, G: primitive goal states
Output: A hierarchical policy for $S^0, ..., S^{L-1}$

UPWARDPASS($S^0, ..., S^{L-1}, A, T, R$)

1 **for** $l = 0$ **to** $L - 1$
2 **for** $i \in S^l$
3 **for** j adjacent to i
4 **if** $l = 0, C^0(i, j) \leftarrow \min_{a \in A} -\frac{R(i,a)}{T(i,a,j)}$
5 **else** $C^l(i, j) \leftarrow$ $\frac{1}{|i|} \sum_{i' \in i} \min_{j' \in j}$[DIJKSTRA($i', j', C^{l-1}$)]
6 **return** C

DOWNWARDPASS($S^0, ..., S^{L-1}, A, T, R, G, C$)

1 **for** $l = L - 1$ **to** 1
2 **for** $i' \in S^l$ contained in $i \in S^{l+1}$
3 **if** $l = L - 1$, $g \leftarrow$ level $L - 1$ goal macro-state
4 **else** $g \leftarrow \pi^{l+1}(i)$
5 $\pi^l(i') \leftarrow$ $\arg\min_{j' \in S^l} C^l(i', j') +$ DIJKSTRA(j', g, C^l)
6 **for** $i \in S^1$
7 $M \leftarrow$ CREATEMDP($i, \pi^1(i), A, T, R, \Delta$)
8 $\pi^0_i \leftarrow$ VALUEITERATION(M)
9 **for** $i' \in S^0$, $\pi^0(i') \leftarrow \pi^0_{\arg\min_{\{i \in S^1 | i' \in i\}} D^1(i)}(i')$
10 **return** π

for the cost of transitioning between two macro-states. We assume that, at the primitive level, any action a taken in state i with the goal of ending in state j does make a transition to j with probability $T(i, a, j)$; but that with the remaining probability mass, it stays in state i. Executing such an action a repeatedly will, in expectation, take $1/T(i, a, j)$ steps to move to state j, each of which costs $-R(i, a)$. We can select whichever action would minimize this cost, yielding the cost estimate shown on line 4 of UPWARDPASS. Once we have level 0 costs, we can solve deterministic shortest path problems to compute costs at all levels.

Next, we execute DOWNWARDPASS to find the hierarchical policy π. At the top levels, we use a deterministic shortest path algorithm to assign the policy. At level 0, rather than using the expected costs to solve a shortest-paths problem, we take the actual transition probabilities over the primitive states into account. In order to do this, we construct an MDP that represents the problem of moving from a macro-state i to a macro-state $\pi^1(i)$. Most of the transition probabilities and reward values have already been defined in the original MDP model. We treat all states in $\pi^1(i)$ as zero-reward absorbing local goal states. We model transitions to states that are neither in $\pi^1(i)$ nor in node i itself as going to a single special *out* state and incurring a fixed, large, negative penalty Δ. We use value iteration to solve for the policy.

The cost of solving at the upper levels is dominated by a quadratic deterministic shortest path algorithm. The time at the bottom level is dominated by value iteration, which Bertsekas (1995) showed is cubic in the size of the state space for stochastic shortest path problems.

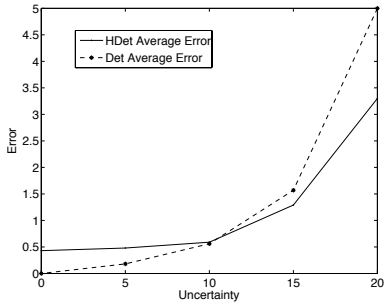

Figure 1: Average deviation from the optimal policy as a function of uncertainty in the grid world domain. Here $x\%$ uncertainty refers to the probability an action transitions to a wrong square. The probability of transitioning to the correct square is $1 - 0.03x$.

Theorem 4: Algorithm 2 has time complexity quadratic in the size of the largest macro-state and the number of $L - 1$ macro-states and cubic in the size of the largest level 1 macro-state.

3.3 Results

We tested the algorithm described above (consisting first of clustering and then of solving), called *HDet* for *hierarchically determinized*, on several different enumerated-states domains, and compared its performance to that of value iteration and HVI (Bakker, Zivkovic, and Krose 2005). HVI originally used spectral clustering, reported as HVI (S); we also tried it with g-connected clustering, reported as HVI (G). We also tried a version of HDet, Det, which does not run the clustering algorithm at all but instead treats each state as its own cluster. Det never solves any MDPs.

We used three experimental domains. *Grid world* is a typical grid world with four actions each with an 85% chance of transitioning to the expected square and a 5% chance of transitioning to each of the other adjacent squares. The *Grid World* had 1040 states and the *Large Grid World* had 62500 states. *Factory* is a version of the common *Builder* factored MDP problem (Dearden and Boutilier 1997), run on the fully enumerated state space of 1024 states. *Mountain Car* is a discretized, randomized version of the Mountain Car domain ((Sutton and Barto 1998), section 8.2) with 1024 states. For full explanations of these domains see (Barry 2009).

We evaluated the policies in each domain by running 1000 simulations of each policy starting from each state in the domain and averaging together the total reward from the simulations to find a policy value for every state in the domain. We report the average deviation of these policy values from the optimal values. Results on the algorithms on each of the domains are shown in Table 1.

Running time vs. accuracy The results show that HDet is substantially faster than value iteration with a modest decrease in the quality of the solution. HDet also substantially outperforms HVI. The closest competitor to HDet is, in fact, Det, the purely deterministic, non-hierarchical, version of HDet. The speed of execution of Det on most of the problems is due to the relatively small size of these problems,

	Grid World		Large Grid World		Factory		Mountain Car	
Algorithm	Run Time (s)	Avg. Dev.	Run Time (s)	Avg. Dev.	Run Time (s)	Avg. Dev.	Run Time (s)	Avg. Dev.
Value Iteration	20.46	0	$> 10^4$	–	25.22	0	83.00	0
HDet	1.41	0.48	74.21	0	2.58	0.49	25.79	4.14
Det	0.19	0.18	94.73	0.2	0.25	0.35	0.51	15.55
HVI (G)	10.66	0.84	$> 10^4$	–	40.72	0.62	78.94	12.94
HVI (S)	24.40	0.66	$> 10^4$	–	81.32	2.36	124.08	236.58

Table 1: Results for three domains. *Run time* gives the total running times, which for HDet and HVI includes clustering time as well as solution time. *Avg. Dev.* is the deviation from the reward of the optimal policy divided by the length of the corresponding path. HVI and value iteration did not converge on the large grid world so we report average deviation from the policy found by HDet, which had the highest value. All algorithms were implemented by us and run on the same computer.

chosen to enable value iteration to terminate in reasonable time. In the larger Grid World problem, Det required more time than HDet. We expect HDet's advantage to increase with the size of the problem.

Similarly, as the non-determinism in the domain increases, we expect the accuracy of both Det and HDet to suffer, but the average deviation of Det increases faster than that of HDet, so that when there is only a 40% chance of ending up in the intended square, the average deviation of Det is close to 5, but that of HDet is closer to 3 (Figure 1). We can control how accurate HDet is by setting the parameters MINCLUST and MAXSIZE. With fewer and larger clusters HDet will be more accurate, but slower.

Thus, when run on enumerated-states MDPs, HDet finds good approximate solutions and has total running times (for clustering and solving) that improve substantially on competing methods. It is important to note that the time taken to create the hierarchy need not be amortized over several problem instances: it pays off on a single instance.

4 Future Work

We are currently adapting this algorithm to work on factored MDPs. A factored MDP is an MDP $M = \langle X, A, T, R, G \rangle$ where X is a set of *state variables*. The state space S of the MDP can be obtained from X; a state of the MDP $i' \in S$ is an assignment to all state variables.

HDet relies on being able to look at every state in the domain, but in a factored MDP that is no longer possible. Therefore, rather than make g-connected macro-states of primitive states, we are developing an algorithm that makes g-connected macro-states of sets of states or "aggregate states". We try to create the aggregate states in such a way that we do not have many of them, but that all states in an aggregate state behave similarly.

We currently have a working implementation of HDet extended to factored domains, FHDet, but are still attempting to characterize the accuracy and running time of the algorithm and optimize the implementation.

References

Bakker, B.; Zivkovic, Z.; and Krose, B. 2005. Hierarchical Dynamic Programming for Robot Path Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3720–3725.

Barry, J.; Kaelbling, L.; and Lozano-Pérez, T. 2010. Hierarchical Solution of Large Markov Decision Processes. Technical report, Massachusetts Institute of Technology.

Barry, J. 2009. Fast Approximate Hierarchical Solution of MDPs. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Belmont, Massachusetts: Athena Scientific.

Bertsekas, D. P. 1995. *Dynamic Programming and Optimal Control*. Belmont, Massachusetts: Athena Scientific.

Dearden, R., and Boutilier, C. 1997. Abstraction and Approximate Decision-Theoretic Planning. *Artificial Intelligence* 89:219–283.

Dietterich, T. G. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In *ICML*, 118–126.

Hauskrecht, M.; Meuleau, N.; Boutilier, C.; Kaelbling, L. P.; and Dean, T. 1998. Hierarchical solution of Markov decision processes using macroactions. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*.

Jonsson, A., and Barto, A. 2006. Causal Graph Based Decomposition of Factored MDPs. *Journal of Machine Learning Research* 7:2259–2301.

Lane, T., and Kaelbling, L. P. 2002. Nearly Deterministic Abstractions of Markov Decision Processes. In *AAAI-02*.

Mahadevan, S. 2008. *Representation Discovery Using Harmonic Analysis*. Morgan and Claypool Publishers.

McGovern, A., and Barto, A. 2001. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. *ICML-01* 361–368.

Mehta, N.; Ray, S.; Tadepalli, P.; and Dietterich, T. 2008. Automatic Discovery and Transfer of MAXQ Hierarchies. In *ICML-08*.

Parr, R., and Russell, S. 1997. Reinforcement learning with hierarchies of machines. In *Neural Information Processing Systems*.

Parr, R. 1998. Flexible decomposition algorithms for weakly coupled markov decision processes. In *Uncertainty in Artificial Intelligence*.

Simsek, O.; Wolfe, A. P.; and Barto, A. G. 2005. Identifying Useful Subgoals in Reinforcement Learning by Graph Partitioning. In *Proceedings of the 22nd International Conference on Machine Learning*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* 112(1-2):181–211.