In Proceedings of the 17th International FLAIRS Conference, pp. 509-514, Miami Beach, FL *© 2004 AAAI*

State Space Reduction For Hierarchical Reinforcement Learning

Mehran Asadi and **Manfred Huber**
Department of Computer Science and Engineering
University of Texas
Arlington, TX 76019
{asadi,huber}@cse.uta.edu

Abstract

This paper provides new techniques for abstracting the state space of a Markov Decision Process (MDP). These techniques extend one of the recent minimization models, known as ϵ-reduction, to construct a partition space that has a smaller number of states than the original MDP. As a result, learning policies on the partition space should be faster than on the original state space. The technique presented here extends reduction to SMDPs by executing a policy instead of a single action, and grouping all states which have a small difference in transition probabilities and reward function under a given policy. When the reward structure is not known, a two-phase method for state aggregation is introduced and a theorem in this paper shows the solvability of tasks using the two-phase method partitions. These partitions can be further refined when the complete structure of reward is available. Simulations of different state spaces show that the policies in both MDP and this representation achieve similar results and the total learning time in partition space in presented approach is much smaller than the total amount of time spent on learning on the original state space.

Introduction

Markov decision processes (MDPs) are useful ways to model stochastic environments, as there are well established algorithms to solve these models. Even though these algorithms find an optimal solution for the model, they suffer from the high time complexity when the number of decision points is large(Parr 1998; Dietterich 2000). To address increasingly complex problems a number of approaches have been used to design state space representations in order to increase the efficiency of learning (Dean Thomas; Kaelbling & Nicholson 1995; Dean & Robert 1997). Here particular features are hand-designed based on the task domain and the capabilities of the learning agent. In autonomous systems, however, this is generally a difficult task since it is hard to anticipate which parts of the underlying physical state are important for the given decision making problem. Moreover, in hierarchical learning approaches the required information might change over time as increasingly competent actions become available. The same can be observed in biological systems where information about all muscle fibers is initially instrumental to generate strategies for coordinated movement. However, as such strategies become established and ready to be used, this low-level information does no longer have to be consciously taken into account. The methods presented here build on the ϵ-reduction technique developed by Dean et al.(Givan & Thomas 1995) to derive representations in the form of state space partitions that ensure that the utility of a policy learned in the reduced state space is within a fixed bound of the optimal policy. The presented methods here extend the ϵ-reduction technique by including policies as actions and thus using it to find approximate SMDP reductions. Furthermore it derives partitions for individual actions and composes them into representations for any given subset of the action space. This is further extended by permitting the definition of two-phase partitioning that is initially reward independent and can later be refined once the reward function is known. In particular the techniques described in the following subsections are to extend ϵ-reduction(Thomas Dean & Leach 1997) by introducing the following methods:

- Temporal abstraction
- Action dependent decomposition
- Two-phase decomposition

Formalism

A Markov decision processes (MDP) is a 4-tuple (S, A, P, R) where S is the set of states, A is a set of actions available in each state, P is a transition probability function that assigns a value $0 \leq p \leq 1$ to each state-action pair, and R is the reward function. A transition function is a map $P : S \times A \times S \rightarrow [0, 1]$ and usually is denoted by $P(s'|s, a)$, which is the probability that executing action a in state s will lead to state s' . Similarly, a reward function is a map $R : S \times A \rightarrow \Re$ and $R(s, a)$ denotes the reward gained by executing action a in state s.
Any policy defines a value function and the Bellman equation (Bellman 1957; Puterman 1994) creates a connection between the value of each state and the value of other states by:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s))V^{\pi}(s')$$

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Previous Work

State space reduction methods use the basic concepts of a MDP such as transition probabilities and reward function to represent a large class of states with a single state of the abstract space.The most important issues that show the generated abstraction is a valid approximate MDP are:

1. The difference between the transition function and reward function in both models has to be a small value.
2. For each policy on the original state space there must exist a policy in the abstract model. And if a state s is not reachable from state s' in the abstract model, then there should not exist a policy that leads from s to s' in the original state space.

SMDPs

One of the approaches in treating temporal abstraction is to use the theory of semi Markov decision processes (SMDPs). The actions in SMDPs take a variable amount of time and are intended to model temporally extended actions, represented as a sequence of primary actions.

Policies: A policy (option) in SMDPs is a triple $o_i = (I_i, \pi_i, \beta_i)$(Boutillier & Hanks 1995), where I_i is an initiation set, $\pi_i : S \times A \longrightarrow [0,1]$ is a primary policy and $\beta_i : S \longrightarrow [0,1]$ is a termination condition. When a policy o_i is executed, actions are chosen according to π_i until the policy terminates stochastically according to β_i. The initiation set and termination condition of a policy limit the range over which the policy needs to be defined and determine its termination. Given any set of multi-step actions, we consider the policy over those actions. In this case we need to generalize the definition of value function. The value of a state s under an SMDP policy π^o is defined as(Boutillier & Goldszmidt 1994):

$$V^{\pi}(s) = E\left[R(s,o) + \sum_{s'} F(s'|s,o)V^{\pi}(s')\right]$$

where

$$F(s'|s,o) = \sum_{k=1}^{\infty} P(s_{t+m} = s' \wedge m = k | s_t = s, o_i)\gamma^k \quad (1)$$

and

$$R(s',o) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|\epsilon(\pi^o, s, t)] \quad (2)$$

r_t denotes the reward at time t and $\epsilon(\pi^o, s, t)$ denotes the event of an action under policy π^o initiated at time t and in state s.

ϵ-reduction Method

Dean et al.(Thomas Dean & Leach 1997) introduced a family of algorithms that take a MDP and a real value ϵ as an input and compute a bounded parameter MDP where each closed interval has a scope less than ϵ. The states in this MDP correspond to blocks of a partition of the state space in which the states in the same block have the same proprieties in terms of transitions and rewards.

Definition 1 *A partition $P = \{B_1, ..., B_n\}$ of the state space of a MDP, M, has the property of ϵ-approximate stochastic bisimulation homogeneity with respect to M for $0 \leq \epsilon \leq 1$ if and only if for each $B_i, B_j \in P$, for each $a \in A$ and for each $s, s' \in B_i$*

$$|R(s,a) - R(s',a)| \leq \epsilon \quad (3)$$

and

$$\left| \sum_{s'' \in B_j} P(s''|s,a) - \sum_{s'' \in B_j} P(s''|s',a) \right| \leq \epsilon \quad (4)$$

Definition 2 *The immediate reward partition is the partition in which two states s, s' are in the same block if they have the same rewards.*

Definition 3 *The block B_i of a partition P is stable(Thomas Dean & Leach 1997) with respect to block B_j if and only if for all actions $a \in A$ and all states $s, s' \in B_i$ equation 4 holds.*

The model reduction algorithm first uses the immediate reward partition as an initial partition and refines them by checking the stability of each block of this partition until there are no unstable blocks left. For example, when block B_i happens to be unstable with respect to block B_j, the block B_i will be replaced by a set of sub-blocks $B_{i_1}, ..., B_{i_k}$ such that each B_{i_m} is a maximal sub-block of B_i that is ϵ-stable with respect to B_j. Once the stable blocks of the partition have been constructed, the transition and reward function between blocks can be defined. The transition of each block by definition is the interval with the bounds of maximum and minimum probabilities of all possible transitions from all states of a block to the states of another block:

$$\hat{P}(B_j|B_i,a) = \left[\min_{s \in B_i} \sum_{s' \in B_j} P(s'|s,a), \max_{s \in B_i} \sum_{s' \in B_j} P(s'|s,a)\right]. \quad (5)$$

And similarly the reward for a block is:

$$\hat{R}(B_j,a) = \left[\min_{s \in B_j} R(s,a), \max_{s \in B_j} R(s,a)\right]. \quad (6)$$

Extension To ϵ-reduction Method

While the ϵ-reduction technique permits the derivation of appropriate state abstractions in the form of state space partitions, it poses several problems when being applied in practice. First, it heavily relies on complete knowledge of the reward structure of the problem. The goal of the original technique is to obtain policies with similar utility. In many practical problems, however, it is more important to achieve the task goal than it is to do so in the optimal way. In other words, correctly representing the connectivity and ensuring the achievability of the task objective is often more important than the precision in the value function. To reflect this, the reduction technique can easily be extended to include separate thresholds ϵ and δ for the transition probabilities and the reward function,respectively. This makes it more

flexible and permits emphasizing task achievement over the utility of the learned policy. The second important step is the capability of including the state abstraction technique into a hierarchical learning scheme. This implies that it should be able to efficiently deal with increasing action spaces that over time include more temporally extended actions in the form of learned policies. To address this, the abstraction method should change the representation as such hierarchical changes are made. To achieve this while still guaranteeing similar bounds on the quality of a policy learned on the reduced state space, the basic technique has to be extended to account for actions that perform multiple transitions on the underlying state space. The final part of this section, discusses the space reduction when the reward function is not available. In this situations, refinement can be done using transition probabilities. This method also shows that when it is necessary to run different tasks in the same environment, refinement by transition probabilities has to be performed only for the first task and can subsequently be augmented by task specific reward refinement. In this way the presented methods can further reduce the time complexity in situations when multiple tasks have to be learned in the same environment.

ϵ, δ-Reduction for SMDP

For a given MDP we construct the policies $o_i = (I_i, \pi_i, \beta_i)$ by defining sub-goals and finding the policies i that lead to sub-goals from each state .The transition probability function $F(s|s', o_i)$ and the reward function $R(s, o_i)$ for this state and policy can be computed with equations 1 and 2. Discount and probabilities are folded here into a single value. As can be seen here, calculation of this property is significantly more complex than in the case of single step actions. However, the transition probability is a pure function of the option and can thus be completely pre-computed at the time at which the policy itself is learned. As a result, only the discounted reward estimate has to be re-computed for each new learning task.The transition and reward criteria for constructing a partition over policies can be refined by:

$$|R(s, o_i) - R(s', o_i)| \leq \epsilon$$

and

$$\sum_{s'' \in B_j} F(s''|s, o_i(s)) - \sum_{s'' \in B_j} F(s''|s', o_i(s)) \leq \delta$$

Action Dependent Decomposition of ϵ, δ-Reduction

Let M be a SMDP with n different options $O = \{o_1, \ldots, o_n\}$ and let $P_1, \ldots, P_n$ be the ϵ, δ-partitions corresponding to each action where $P_i = \{B_i^1, \ldots, B_i^{m_i}\}$ for $i \in W = \{i | o_i \in O\}$. Define $\Phi = P1 \times P2 \times \ldots \times P_n$ as the cross product of all partitions. Each element of Φ has the form $\phi_j = (B_1^{\sigma_1(j)}, \ldots, B_n^{\sigma_n(j)})$ where σ_i is a function with domain $|\Phi|$ and range $1, \ldots, m_i$. Each element $\phi_j \in \Phi$ corresponds to a $\tilde{B_j} = \cap_{i \in A} B_i^{\sigma_i(j)}$. Given a particular subset of options, a partition for the learning task can now be derived as the set of all non-empty blocks resulting form the intersection of the subsets for the participating options. A block in the resulting partition can therefore be represented by a vector over the options involved, where each entry indicates the index of the block within the corresponding single action partition. Once the initial blocks are constructed by the above algorithm, these blocks will be refined until they are stable according to ϵ, δ-method. Changes in the action set therefore do not require a recalculation of the individual partitions but only changes in the length of the vectors representing new states and a recalculation of the final refinement step is required. This means that changes in the action set can be performed efficiently and a simple mechanism can be provided to use the previously learned value function even beyond the change of actions and to use it as a starting point for subsequent additional learning. This is particularly important if actions are added over time to permit refinement of the initially learned policy by permitting finer-grained decisions.

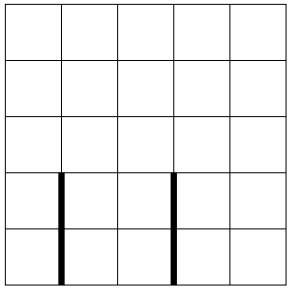

Figure 1: Grid world for example

A Simple Example

In this example we assume a grid world with a mobile robot which can perform four primitive deterministic actions: left, right, up and down. Rewards for actions that lead the agent to another cell are assumed to be -1. In order to construct an option we define a policy with each action. The termination condition is hitting the wall and the policy repeats each action until it terminates. Figure 1 shows this scenario. Figures 2.a through 2.d show the possible partitions for the four options. Let B_i^j be the block j for partition i derived by action o_i. Then the cross product of these blocks is a vector containing all possible combination of these blocks:

$$\Phi = \{B_1^1, B_1^2\} \times \{B_2^1, B_2^2\} \times \{B_3^1, B_3^2\} \times \{B_4^1, B_4^2\}$$

The intersection partition has the elements:

$$\begin{aligned} \tilde{B_1} &= B_1^1 \cap B_4^1 \cap B_2^1 \cap B_3^1 \\ \tilde{B_2} &= B_1^2 \cap B_4^1 \cap B_2^1 \cap B_3^1 \\ &\vdots \end{aligned}$$

Figure 3 illustrates the intersection of the partitions. These blocks form the initial blocks for the reduction technique. The result of refinement is illustrated in Figure 4.While performing an action on each state, the result would be another block instead of a state so each block of Figure 4 can be considered a single state in the resulting BMDP.

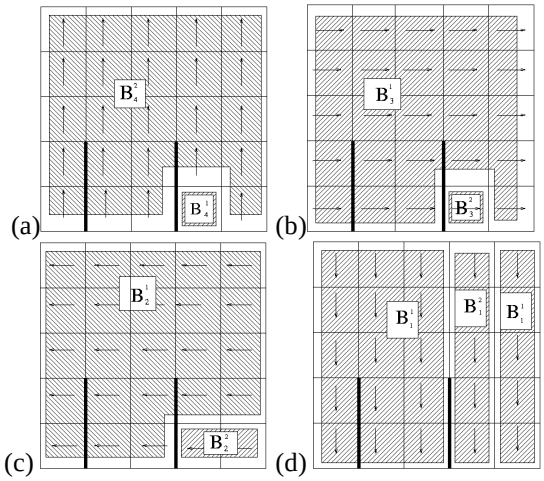

Figure 2: Blocks for options a)Up b)Right c)Left d)Down

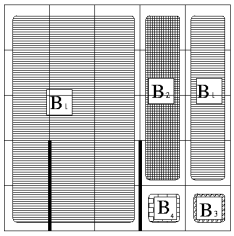
Figure 3: Intersection of Blocks

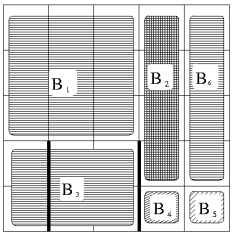
Figure 4: Final Blocks

Two-Phase Partitions

Environments usually do not provide all the necessary information, and the agent needs to determine these details by itself. For example, it is common that an agent does not have full information of the reward structure. In these situations, constructing the immediate reward partition is not possible and the partitions for reduction have to be determined differently, the algorithm introduced in this section drives partitions in two different phases. This two-phase method constructs the initial blocks by distinguishing terminal states for available actions from non-terminal states and refines them using the transition probabilities. (Asadi 2003).

Definition 4 *A subset C of the state space S , is called a terminal set under action a if $P(s|s',a) = 0$ for all $s' \in C$ and $s \notin C$.*

Definition 5 *$P^{(n)}(s|s',a)$ denotes the probability of first visit to state s from state s'. That is,*

$$P^{(n)}(s|s',a) = P(s_{n+k} = s | s_{n+k-1} \neq s, ..., s_k \neq s, a, s').$$

Definition 6 *For fixed states s and s' , let $F^*(s|s',a) = \sum_{n=1}^{\infty} P^{(n)}(s|s',a)$. $F^*(s|s',a)$ is the probability of ever visiting state s from state s' .*

Proposition 1 *A state s belongs to a terminal set with respect to action a if $F^*(s|s',a) = 1$.*

Proposition 1 shows a direct way to find the terminal sets, i.e. the termination condition for each option. Once the terminal sets are constructed, the state space can be partitioned by transition probabilities. In situations where the reward information is not available the two-phase method can be used to learn an initial policy before the complete reward structure has been learned. While the utility of the policy learned prior to reward refinement might not be within a fixed bound of the one for the optimal policy that would be learned on the complete state space, it can be shown that for a certain subset of tasks the initial policy is able to achieve the task objective. After the reward structure has been determined, however, the final policy for all tasks will have the ϵ, δ property and thus be within a fixed bound of the optimal policy on the full state space. Being able to compute an initial policy on the non reward refined state space here permits the system to determine a policy that is useful for the task prior to establishing a more optimal one after reward refinement.

Theorem 1 *For any policy π for which the goal G can be represented as a conjunction of terminal sets (sub-goals) of the available actions in the original MDP M, there is a policy π_Φ in the reduced MDP,M_Φ, that achieves G if for each state in M for which there exists a path to G , there exists such a path for which $F(s_{t+1}|s_t, \pi(s_t)) > \delta$.*

Proof The blocks of partition $\Phi = \{B_1, ..., B_n\}$ have the following property

$$\left| \sum_{s \in B_j} F(s|s_1, o_i(s_1)) - \sum_{s \in B_j} F(s|s_2, o_i(s_2)) \right| \leq \delta$$

For every policy π that fulfills the requirements of the proposition, there exists a policy π_Φ in partition space such that for each $n \in \aleph$, if there is a path of length n from state s_0 to a goal state G, under policy π, then there is a path for block B_{s_0} containing s_0 to block B_G containing G, under policy π_Φ.

Case $k = 1$: if $F(G|s_0, \pi(s_0)) > \delta$ then by condition (9) for all $s \in B_{s_0}$,

$$\left| \sum_{s' \in B_G} F(s'|s_0, \pi(s_0)) - \sum_{s' \in B_G} F(s'|s, \pi(s_0)) \right| \leq \delta$$

thus $\forall s \in B_{s_0}$ then $F(G|s, \pi(s_0)) > F(G|s_0, \pi(s_0)) - \delta > 0$. Define policy π_Φ such that $\pi_\Phi(B_{s_0}) = \pi(s0)$, then $F(B_G|B_{s_0}, \pi_\Phi(B_{s_0})) > 0$.

Case $k = n - 1$: Assume for each path of length less than or equal to $n - 1$ that reaches state G from s_0 under policy π, there is a path under policy π_Φ in the partition space.

Case $k = n$: Each path that reaches with G from s_0 under policy π in n steps contains a path with $n - 1$ steps, that reaches G from s_1 under policy π. By induction hypothesis, there is a policy π_Φ that leads to B_G from B_{s_1} . Now if s_0 is an element of $B_{s_n} \cup B_{s_{n-1}} \cup, \ldots, \cup B_{s_1}$, the blocks already chosen by paths with length less than or equal $n - 1$, then there is a policy π_Φ that leads to B_G from B_{s_0} under policy π_Φ and the policy $\pi_\Phi(B_{s_0})$ is already defined. But if $s_0 \notin B_{s_n} \cup B_{s_{n-1}} \cup, ..., \cup B_{s_1}$ then by induction hypothesis it has only to be shown that there is a policy π_Φ that fulfills the induction hypothesis and which leads from B_{s_0} to B_{s_1} such that $F(B_{s_1}|B_{s_0}, \pi(s_0)) > 0$. By condition (9) $\forall s1, s2 \in B_{s_0}$ we have $|\sum_{s' \in B_G} F(s|s_0, \pi_i(s_0)) - \sum_{s' \in B_G} F(s'|s, \pi_i(s_0))| \leq \delta$, thus

$$F(B_{s_1}|B_{s_0}, \pi_\Phi(B_{s_0})) = \sum_{s' \in B_{s_1}} F(s'|s, \pi(s_0))$$

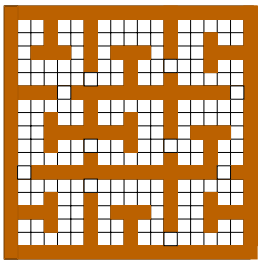

Figure 5: The pattern for experiment 1

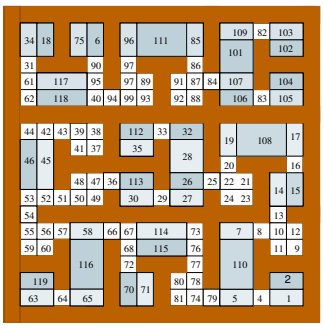

Figure 6: Final block for experiment 1

$$\geq F(s'|s_0, \pi(s_0)) - \delta > 0$$

Therefore the proposition holds and thus for every policy with the goal defined as in the proposition there is a policy that achieves the goal in the two-phase partition.□

Experimental Results

Experiment 1 In order to compare the model minimization methods that are introduced in this paper, several state spaces with different sizes have been examined with this algorithm. These examples follow the pattern illustrated in Figure 5. The underlying actions are left, right, up and down and they are successful with the probability 0.9 and return to the same state with the probability 0.1. In this experiment the primitive actions are extended by repeating them until they hit a wall. Figure 6 illustrates the final blocks of the partition in this experiment and Figure 7 and 8, compare the running time of single phase reduction and two-phase reduction. This comparison shows that the two phase method takes longer than the ϵ-reductions but the reward independent refinement part is faster than refinement in ϵ-reductions and thus the two-phase method is more efficient if the reward refinement in done beforehand, as indicated in the following experiment.

Experiment 2 This experiment has been performed in order to compare the total partitioning time after the first trial. In order to do so, the goal state has been situated in different positions and the run time of ϵ-reduction algorithm and the two-phase algorithm are investigated.

This environment consists of three grid worlds, each of these grid worlds consists of different rooms. The termination states of the available actions are illustrated in black in Figure 9. The actions for each state are multi-step actions. These actions terminate when they reach a sub-goal in the same room.This experiment shows that, even though the run time of ϵ-reduction algorithm is lower than the run time of the reward independent algorithm, after the first trial the reward independent algorithm is much faster, as the transition partitioning for this algorithm is already done in the first trial and transition partitioning is not necessary after it is performed once. Figures 10 and 11 show the difference in run times for 6 different trials. For the first trial the situation is similar to the previous experiment and the total run time in ϵ-reduction method is smaller than the total run time of the two-phase method. Figure 12 illustrates the different between policies in this experiment.

When a different goal is located in the environment, the two-phase method does not need to refine the state space with the transition, as it has been done for the first task. On the other hand the ϵ-reduction method needs to redefine the initial blocks for each task. As a result the total run time after the first task in the two-phase method is significantly smaller than the run time of ϵ-reduction method.

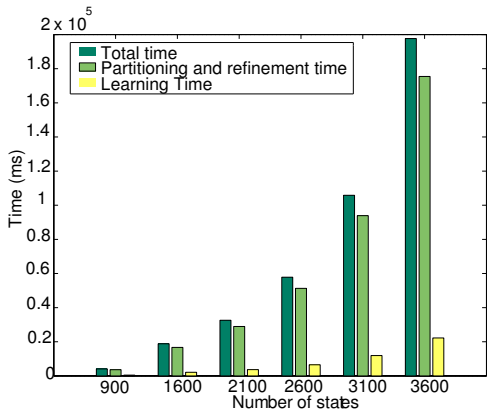

Figure 7: Run time for ϵ-reduction Method

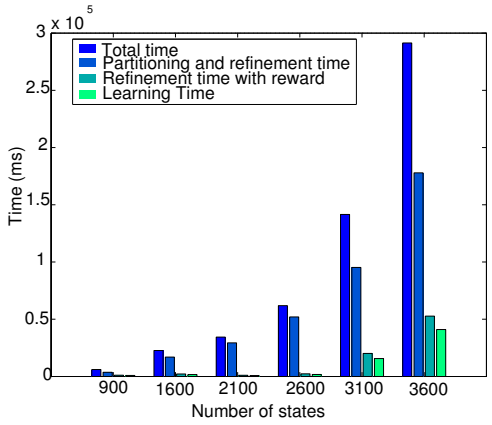

Figure 8: Run time for two-phase method

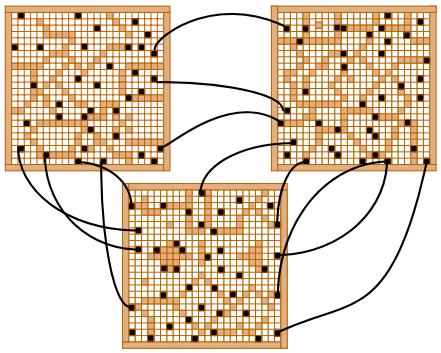

Figure 9: Scenario of the experiment 2

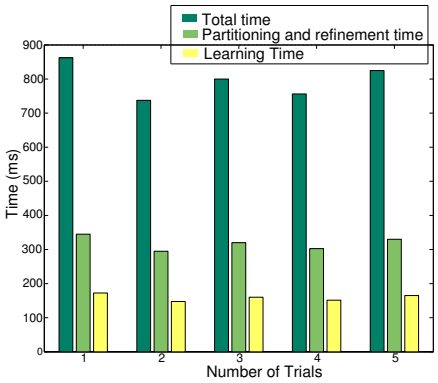

Figure 10: Run time of ϵ-reduction for 5 trials

Conclusion

The results of the techniques described in this paper show that even though the ϵ-reduction method introduces a fine way of partitioning the state space, it can be improved significantly by using temporal abstraction. While the approach introduced here extends the applicability of reinforcement learning techniques by providing state space abstractions that permit more complex tasks to be learned, there are situations in which the reward information is not known. This paper provides a solution for these situations by considering the terminal states and no-terminal state as initial blocks instead of the immediate reward block, and proves that the partitioned space is solvable. The comparison between running different tasks in the same environment shows the significant reduction of time complexity in the two-phase method.

References

Asadi, M. 2003. *State Space Reduction For Hierarchical Policy Formation. Masters thesis*. University of Texas, Arlington.

Bellman, R. 1957. *Dynamic programming*. Princton University Press.

Boutillier, Craig; Dearden, R., and Goldszmidt, M. 1994. Exploiting structure in policy construction. *In Proceedings IJCAI 14*. IJCAII:1104–1111.

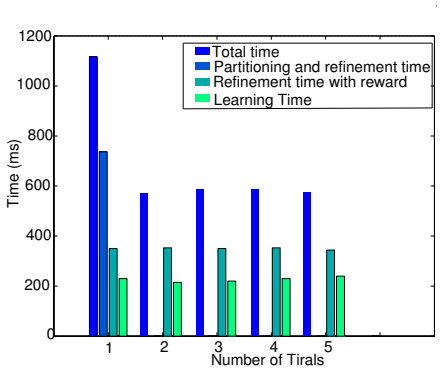

Figure 11: Run time of two-phase method for 5 trials

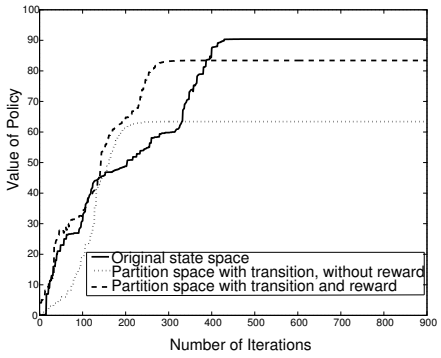

Figure 12: Learning curve for experiment 2

Boutillier, Craig; Dean, T., and Hanks, S. 1995. Planning under uncertainty: Structural assumptions and computational leverage. *In Proceeding of Third European Workshop on Planning*.

Dean, T., and Robert, G. 1997. Model minimization in markov decision processes. *In Proceeding AAAI-97* 76.

Dean Thomas; Kaelbling, Leslie; Kirman, j., and Nicholson, A. 1995. Planning under time constraints in stochastic domains. *Artifitial intelligence* 76:35–74.

Dietterich, T. G. 2000. An overview of maxq hierarchical reinforcement learning. *Lecture Notes in Artificial Intelligence, In Proceeding of 4th International Symposium, SARA* 1864:26–44.

Givan, Robert; Leach, S., and Thomas, D. 1995. Bounded parameter markov decision processes. *Technical Report CS-97-05 Brown university* 35–74.

Parr, R. E. 1998. *Learning for Markov Decision Processes. Doctoral dissertation*. University of California, Berkeley.

Puterman, M. L. 1994. *Markov decision processes*. John Wiley and Sons.

Thomas Dean, R. G., and Leach, S. 1997. Model reducion techniques for computing approximately optimal solution for markov decision process. *In Proceedings of Thirteenth Conference on Uncertainty in Artigicial Intelligence*.