A Unifying Framework for Temporal Abstraction in Stochastic Processes

Ronald Parr
parr@cs.berkeley.edu
Computer Science Division
University of California
Berkeley, CA 94720

Abstract

This paper presents a framework for unifying the large and growing body of literature that deals with what broadly can be defined as temporal abstraction in Markov Decision Processes (MDPs). MDPs provide an appealing formal framework for modeling a large variety of stochastic problems. The main drawback of this approach is that a requirement of the formal model, i.e., the Markov property, typically forces very fine granularity models which, in many cases, can be intractably large. Numerous approaches have been developed to help manage large state spaces by introducing actions at varying levels of granularity, including hierarchical decomposition, multi-layer control, and macro-actions. This paper extracts and generalizes a common thread that runs through all of these methods: the transformation of a policy defined over a region of the state space into a temporally abstract action in a new semi-Markov decision process (SMDP). Convergence theorems are presented for SMDP reinforcement learning and an asynchronous form of policy iteration that exploits this transformation. The existing methods for temporal abstraction are divided into three categories which are analyzed within this framework.

1 Introduction

The representation method of choice for sequential decision problems that involve uncertainty is the Markov decision process (MDP). This representation is important because algorithms are available that determine the optimal policy, i.e., optimal assignment of actions to states, in run time that is polynomial in the number of states in the process. Reinforcement learning algorithms also can be shown to converge to the optimal policy for MDPs.

The difficulty for Artificial Intelligence lies in the requirements of the formal model: states must obey the Markov property, which means that the probability distribution over next states given the current state and action must be independent of any previous states visited or actions taken. Compliance with this requirement tends to produce models that are intractably large, while failure to comply can lead to a loss of performance guarantees for standard MDP algorithms.

Many have looked to the success of hierarchical methods (Tate 1977) in classical planning for inspiration in dealing with large MDP state spaces. Two types of abstraction methods have been developed: *state space abstraction* (Boutilier & Dearden 1994; Dean & Givan 1997) and *temporal abstraction* (Singh 1992a; Lin 1993; Sutton 1995; Dean & Lin 1995; Parr 1996; Precup, Sutton, & Singh 1997; Parr & Russell 1998). State space abstraction aims to compress the state space by grouping similar states together, while temporal abstraction aims to compress the state space by introducing more complex actions that essentially compress time. Some approaches combine both types of abstraction (Dayan & Hinton 1993; Dietterich 1997).

This paper focuses on temporal abstraction and extracts a common, underlying principle behind all of the temporal abstraction methods mentioned above, unifying techniques applied under different circumstances and with different goals. Some authors sought on-line solutions while others sought off-line solutions; some wanted approximate or satisficing solutions while others wanted to find the optimal solution more quickly. This paper shows that each of these methods contains the same basic step, the transformation of a policy defined over a region of the MDP into an *abstract* action in a more general type of stochastic decision process called a semi-Markov decision process (SMDP). This provides a deeper understanding of this broad family of methods and and permits the use of general theorems about SMDPs to analyze their convergence and optimality properties. The main differences among the methods are in how the abstract actions are selected and in how they are used to produce an overall (global) policy. This paper identifies three different ways in which temporal abstractions are used and discusses the properties of and relationships between these methods.

Several new theoretical results for SMDPs are presented in this paper. The convergence of Q-learning for SMDPs is established and a new iterative abstraction method that generalizes over previous off-line abstraction algorithms is presented. This paper presents and proves the correctness of on-line and off-line methods for converting policies to abstract actions.

The next section contains a brief review of MDPs and the following section proceeds with a discussion of the less well-known SMDP framework. Section 4 explains the transformation that turns a policy defined over a region of the space into an abstract action in an SMDP and Section 5 describes the three ways of applying abstract actions and interprets the existing work in this field using the terminology developed here. The paper concludes with a discussion section and directions for future research.

2 Markov Decision Processes

This section assumes some familiarity with the basic concepts of MDPs. (see, e.g., Puterman (1994)) To review, an MDP is a 4-tuple, $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ where $\mathcal{S}$ is a set of *states*, $\mathcal{A}$ is a set of *actions*, $\mathcal{T}$ is a *transition model* mapping $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ into probabilities in $[0, 1]$, and $\mathcal{R}$ is a *reward function* mapping $\mathcal{S} \times \mathcal{A}$ into real-valued rewards. Algorithms for solving MDPs can return a *policy* π that maps from $\mathcal{S}$ to $\mathcal{A}$, a real-valued *value function* V on states, or a real-valued Q-function on state-action pairs. In this paper, we focus on infinite-horizon MDPs with a discount factor β. The aim is to find an optimal policy π^* (or, equivalently, V^* or Q^*) that maximizes the expected discounted total reward of the agent.

The basic algorithms for MDPs all use some form of the Bellman equation (Bellman 1957):

$$V^*(s) = \max_a \mathcal{R}(s, a) + \beta \sum_{s'} \mathcal{T}(s, a, s') V^*(s')].$$

where β is a discount factor on $(0, 1)$. [1] When the Bellman equation is satisfied, the maximizing action for each state is the optimal action.

Value iteration uses successive approximations to achieve V^*. Starting with a random value function V^0, the assignment:

$$V^{t+1}(s) = \max_a \mathcal{R}(s, a) + \beta \sum_{s'} \mathcal{T}(s, a, s') V^t(s').$$

is applied synchronously across all states in $\mathcal{S}$. This is repeated until the state values converge to within some bound of V^*.

Policy Iteration uses the fact that for a fixed policy, π_1, the value equations are linear:

$$V_{\pi_1}(s) = \mathcal{R}(s, a) + \beta \sum_{s'} \mathcal{T}(s, \pi_1(s), s') V^{\pi_1}(s').$$

These can be solved to determine, V_{π_1}, the value of following π_1 from any state. V_{π_1} is plugged in to the Bellman equation and the optimizing action is used to produce a new, better policy, π_2 This process is repeated until the policy cannot be improved, yielding π^*. Linear programming also can be used to find the optimal policy.

An interesting recent result removes the synchrony requirement of value iteration and unifies value iteration and policy iteration. *Asynchronous policy iteration* (API) (Singh & Gullapalli 1993; Bertsekas & Tsitsiklis 1996) maintains a policy, π, and at each phase of the algorithm, a subset, $G \subseteq S$ is selected and for each $s \in G$ one or both of the following operations are performed:

$$V(s) \leftarrow \mathcal{R}(s, \pi(s)) + \beta \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') V(s)$$

or

$$\pi(s) \leftarrow \arg\max_a \mathcal{R}(s, a) + \beta \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') V(s)$$

in any order.

As long as the scheme that is used for determining G ensures that every state has its policy and value updated infinitely often and the initial value function is pessimistic, API will produce V^* and π^*. Note that an infinite number of value updates in API is equivalent to an indirect solution (e.g., the iterative Gauss-Seidel solution) of the system of equations used to determine the value function for policy iteration.

Q-learning can be used to determine an optimal policy online by performing incremental updates to Q functions defined over state-action pairs. On transition from state s to state s' under action a and with reward r:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[\beta V(s') + r - Q(s, a)],$$

where α is a learning rate and $V(s) = \max_a Q(s, a)$.

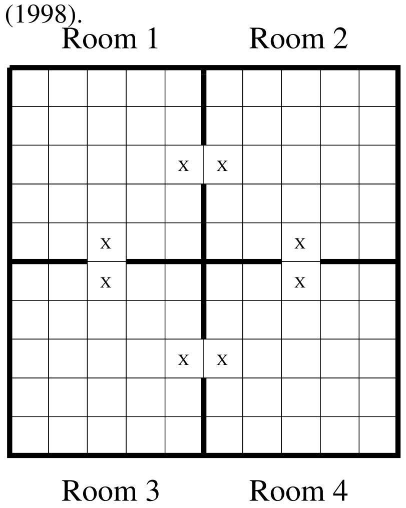
As an example, consider a simple model with a four-room topology as shown in Figure 1. Suppose that room i contains a state with reward r_i and that the process enters an absorbing state after one of the reward states is reached. Differences between the reward values can result in a range of different behaviors. For example, if all of the rewards are the same, the optimal policy will move to the closest reward. On the other hand if r_4 is much larger than the other rewards, then the optimal policy will bypass local rewards to achieve r_4. The tradeoff between local rewards and distant rewards will be determined by the stochasticity of the environment and the discount factor, as these will together determine the time required to get from one part of the environment to another and the opportunity cost of this time.

3 Semi-Markov Decision Processes

Semi-Markov decision processes (SMDPs) are less-studied than MDPs, but their properties are well-documented (Puterman 1994) and they have been used to solve some practical robot navigation problems (Mahadevan, Khaleeli, & Marchalleck 1997). They are a simple generalization of MDPs to model actions as complex events that can take place over variable amounts of time. This can arise quite naturally in the example of the previous section. Suppose that a robot navigating in this domain has an unreliable actuator that controls its motor. If the actuator starts the robot with probability p for each try, then the probability that a movement takes time t is $(1 - p)^{t-1}p$ This introduces a simple form of non-determinism in the time required to make a transition and creates an effective discount factor of $\sum_{t=1}^{\infty}(1-p)^{t-1}p\beta^t$, which the time penalty incurred while the robot is waiting for the motor to start.

[1] Generalizing to the case where β is on $(0, 1]$ is possible if we assume that all policies are guaranteed to reach an absorbing state. This straightforward extension is omitted to keep the presentation of the main concepts here as simple as possible.

Figure 1: An MDP divided into 4 rooms. X indicates states that connect the rooms. Similar representations are used in Precup, Sutton and Singh, (1997) and Hauskrecht et alia (1998).



More generally, time could be a continuous parameter with a distribution that depends on the initial state, the ultimate destination of the transition, and the action taken. Call this distribution, $F(t|s,a,s')$ with corresponding density function, $f(t|s,a,s')$. In the fullest generality, there may be a reward rate, $r(s,a)$ that defines the rate at which rewards are received on transitions. This yields the following form for the Bellman equation:

$$V^*(s) = \max_a \int_0^\infty r(s,a) + \beta^t \sum_{s'} f(t|s,a,s')\mathcal{T}(s,a,s')V^*(s')dt$$

While this representation may initially seem unwieldy, it turns out that the effects of time can be expressed using any of several equivalent representations:

1. A probability distribution over next states and a probability distribution over time for every state-action pair.
2. A probability distribution over next states and a discount factor, $\beta(s,a)$, indexed by state and action.
3. A linear function for each s and a describing the value of state s given that action a is taken in s, expressed as $Q(s,a) = \sum_{s'} c_{s,a,s'} V(s') + k_{(s,a)}$, where $0 < \sum_{s'} c_{s,a,s'} < 1$ and $c_{s,a,s'} \geq 0$ for all s, a and s'.

To achieve 2 from 1, observe as in Bradtke and Duff (1995), that time can be integrated out of the continuous form of the Bellman equation to produce $R(s,a) = \int_0^\infty r(s,a)dt$ and $\beta(s,a) = \sum_{s'} \int_0^\infty \mathcal{T}(s,a,s')\beta^t dt$

To achieve 3 from 2, simply write out the linear Q-functions for each state:

$$Q^*(s,a) = R(s,a) + \sum_{s' \in S} \mathcal{T}(s,a,s') + \beta(s,a)V^*(s')$$

To achieve 1 from 3, select a single discount rate by choosing $\beta_{\max} = \max_{s,a} \sum_{s'} c_{s,a,s'}$ and define a time distribution function, $F(t|s,a)$, for each state-action pair as a step function with a jump to 1 at $t = \frac{\log \beta(s,a)}{\log \beta_{\max}}$ (Parr 1998b).

Most importantly, all of the standard off-line MDP algorithms — value iteration, policy iteration, and linear programming — can be modified to work with an SMDP representation of actions. Moreover, Bradtke and Duff (1995) have proposed a reinforcement learning algorithm for SMDPs by modifying the standard Q-learning rule as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\rho + \beta^t V(s') - Q(s,a))$$

where ρ is the accumulated, discounted reward received for the duration of action a.

Theorem 1 *With a suitable learning rate and exploration strategy, SMDP Q-learning converges with probability* 1.

Proof (Sketch) The proof follows the proof of convergence for regular Q-learning in (Jaakkola, Jordan, & Singh 1994). The difference is the presence of t, time, in the update rule. When this marginalized out, convergence follows from the convergence of asynchronous value iteration for SMDPs (Parr 1998b). ■

These results provide the foundations that allow the transformation of policies to actions in a principled manner that preserves performance guarantees.

4 From policies to actions

Suppose that we have one or more heuristic control strategies for room 2 of Figure 1 and we would like to use these either to find an approximately optimal policy for this domain or as a means of bootstrapping an optimal solution. What are our options for representing and manipulating these types of policies? It appears that the most general representation of a fixed policy that maintains computational tractability is a (hierarchical) closed-loop, stochastic finite state controller (Parr 1996; Precup, Sutton, & Singh 1997; Parr & Russell 1998). The key components of this representation are

- An initial state
- A stochastic function determining the next machine state as a function of features of the current environment state
- State outputs that specify environment actions
- A stop state

Hierarchy is achieved by allowing machines to call others as subroutines (Parr & Russell 1998). Clearly, this representation subsumes the standard state-action mapping representation. It also subsumes "full β models" (Sutton 1995), which are essentially policies with stochastic, state-dependent termination conditions. This is achieved with a machine by adding stochastic transitions to the stop state.

Theorem 2 *The expected, discounted return for executing a policy, π, that can be expressed as a closed-loop, stochastic finite state controller, will be a linear function of the values of states in which π terminates and this function will be a valid SMDP action description.*

Proof (Sketch)

1. Create a new process that mirrors the state space and dynamics of the original MDP.
2. Introduce a new set of states representing the cross-product of the environment states and the policy machine states. Note that there will be no action choices in these states; the action is fixed by the output of the machine component of the state.
3. Replace transitions to cross-product states where the machine component is a stop state with transitions back to the regular states.
4. Introduce an action in the new process that switches to the cross-product states to execute the policy.

We now write out the Bellman equations for the cross-product states. Since the actions are fixed in these states, we can express them as a system of linear equations that is sorted so that the cross-product states appear before any of the regular states:

$$\left(\begin{array}{ccc|ccc|c} c_{11} & c_{12} & \dots & v_{11} & v_{12} & \dots & k_1 \\ c_{21} & c_{22} & \dots & v_{21} & v_{22} & \dots & k_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & k_3 \end{array}\right)$$

where row i comes from the Bellman equation of cross-product state i, c_{ij} comes from the coefficient of *cross-product* state j, v_{ij} comes from the coefficient of *regular* state j and k_i comes from the constant in the Bellman equation, i.e., the reward.

We then solve this system, which results in the following form:

$$\left(\begin{array}{ccc|c} 1 & 0 & \dots & f_1 \\ 0 & 1 & \dots & f_2 \\ \vdots & \vdots & \ddots & \vdots \end{array}\right)$$

where the functions $f_1 \dots f_n$ are linear functions that express the values of the cross-product states in terms of the regular states in which π terminates. Any state that can initiate π can express the Q value of launching π as a linear combination of $f_1 \dots f_n$. It is easy to show that the sum of the coefficients will be on $(0, 1)$ if the original MDP has a discount factor on $(0, 1)$. As established in the previous section, this representation is sufficient for the description of an SMDP action.■

In the four-room example, this theorem shows that the effects of a policy defined over room 2 can be summarized with an SMDP action defined at the states in rooms 1 and 4 that connect to room 2.

A useful property of the Q-learning rule for SMDPs is that it permits the use of abstract actions without requiring the explicit transformation described above. This is done using the following *cumulative reward and discount rules*: When abstract action, π, is launched, initialize ρ to the immediate reward, and β_c to β, the discount factor the original MDP. After each step, if π does not terminate:

$$\rho \leftarrow \rho + \beta_c r$$

where r is the immediate reward for the current step and

$$\beta_c \leftarrow \beta\beta_c.$$

If π does terminate, then the standard Q-learning update is applied:

$$Q(s, \pi) \leftarrow Q(s, \pi) + \alpha[\rho + \beta_c V(s') - Q(s, \pi)].$$

Theorem 3 *With the cumulative reward and discount rules, and suitable learning rate and exploration strategy, Q-learning will learn the expected discounted return of applying any policy, π, expressible as stochastic, closed loop finite state controller.*

Proof Observe that the expected value propagated back under the cumulative reward and discount rules equals the right hand side of the Bellman equation for the SMDP action corresponding to π. This makes the update rule an instance of SMDP Q-learning for the SMDP that captures the dynamics of π as constructed in Theorem 2. ■

5 Using temporal abstractions

With a common framework to understand the mechanics of temporal abstraction we can understand more clearly the differences among various temporal abstraction methods. There appear to be three main approaches to the use of abstract actions. *Reductive* methods rely upon a heuristic or external source to provide the policies needed to produce abstract actions. Several abstract actions may be generated for the same region of the state space and an SMDP is solved to determine the optimal combination of abstract actions. *Iterative* methods use abstract actions as a means of reducing run time or learning time, but may abandon the initial set of abstract actions, or iteratively refine them on the path to a globally optimal solution. *Concurrent* methods aim to reduce the effort required to achieve optimality by combining a predefined set of abstract actions with concrete actions.

5.1 Reductive methods

In our four-room domain, we can construct a simple example of the reductive use of a temporal abstraction. Suppose, as in the previous section, that we have a heuristically determined policy, π, which can be initiated from any of the states connecting room 2 with rooms 1 and 4 for room 2. If we restrict the set of global policies that we are willing to consider for this problem to just those that are consistent with π, then we can find the optimal policy consistent with π by solving a *smaller* SMDP:

Theorem 4 *If policies of some region G of an MDP are constrained to be chosen from Π, then there is an SMDP with $|G|$ fewer states than the original MDP and at most $|\Pi|$ more actions that makes the optimal choice of policies for G from Π.*

Proof Create a new SMDP that implements the policies in Π. For each state bordering C, there will be $|\Pi|$ abstract actions, one for each policy in Π. Remove the states corresponding to the *original* states of C from the new SMDP. The new SMDP will model the original process with the restriction that only policies in Π may be executed in C. ■

This basic result, which was first discovered by Forestier and Varaiya (Forestier & Varaiya 1978) and rediscovered by others many times thereafter, captures the essence of the reductive use of temporal abstractions. By reasoning exclusively about high-level actions in one region of the state space, low-level actions are removed from consideration. This result generalizes to multiple policy sets defined over multiple regions. For example, a cache of policies could be defined for each room, and a high-level SMDP would be solved to determine the optimal assignment of policies to each room from its respective cache.

One of the interesting features of the solution to the SMDP in the above proof is that it allows different states bordering G to select different policies over G. Thus, a policy which is optimal and stationary at the high level, may be sub-optimal and non-stationary at the low level. This should be seen as a natural consequence of abstraction that results when the optimal solution to the low-level problem is not expressible in the language of the high-level problem.

This reductive use of abstract actions has appeared many times in the literature. The earliest was in Forestier and Varaiya (1978) where the topic was approached from the perspective of controlling a plant. Abstract actions arose as expert-provided control laws designed to keep the plant out of dangerous states. Thus, the solution to the high-level problem was a switching procedure that switched off between expert control programs when a dangerous state was encountered.

Abstract actions are used reductively in a reinforcement learning context by Lin (1993) from the perspective of robot control. In Lin's approach, the abstract actions defined by policies that achieved subtasks that were deemed *a priori* to be relevant to the global task. Singh (Singh 1992b) explored the relationship between subtasks and high-level tasks in a more formal setting, describing the conditions under which subtask solutions could be combined to produce a globally optimal solution.

The HAM language (Parr & Russell 1998), which is intended for both reinforcement learning and off-line use, generalized the reductive framework by introducing language for expressing policies that permits subroutines and incomplete policy specifications. The HAM language also introduced an agent-centered view of temporal abstraction by defining the abstract actions in terms of procedure calls that could be initiated in response to the agent's sensor stimuli.

The convergence and optimality properties of these methods, which were originally established independently, and without reference to the SMDP formalism, all follow as a direct consequence and optimality properties of SMDP algorithms. It is important to keep in mind, however, that the quality of the resulting solution will be only as good as the quality of the abstract actions used at the high level. Moreover, the complexity of the high-level problem could be greater than that of the original problem if the number of abstract actions is too large.

Feudal RL (Dayan & Hinton 1993) and MAX-Q (Dietterich 1997) also employ temporal abstraction, but combine it with other forms of abstraction. The temporal aspect of these methods can be analyzed within this framework, but a general understanding of the relationship between temporal abstract and other forms of abstraction is an important and ongoing area of research.

5.2 Iterative methods

The natural step to overcome the dependence of reductive methods upon the quality of the set of abstract actions is the introduction of some kind of iterative refinement step that improves the abstract actions if the global policy is not of high enough quality.

Suppose that we begin as in the previous subsection, by using a set of policies defined over rooms as abstract actions. One way to generate an initial set of policies is to make some guesses about the values of the 8 states that connect the rooms. These guesses can be used to determine an initial set of policies for each room by solving a series of separate MDPs. For example, in room 2, we would iterate over our guesses for the values of the connecting states. For each guess, we would find a policy for room 2 by disconnecting the room from the rest of model and finding the optimal policy for the room under the assumption that the adjacent states in rooms 1 and 4 are absorbing states with fixed values corresponding to our guesses.

The solution to each of these smaller MDPs would form the basis of an abstract action in a reduced SMDP. When this is done for each room, the result is an SMDP with just 8 states, the 8 connecting states (marked with X in the figure), and a set of abstract actions corresponding to policies defined over the rooms. The optimal assignment of policies to rooms could be determined using the methods of the previous subsection.

If the global solution produced by the reductive use of the room policies is not good enough, we can attempt to improve the set of policies by using the values of the states in the reduced SMDP to guide creation of a new set of abstract actions. In the four-room example, we would re-run the steps used to determine the initial policies using the values from the solution to the SMDP as updated guesses for the values of the connecting states.

Dean and Lin (1995) use a very similar approach to the iterative refinement of abstract actions based upon the Dantzig-Wolfe decomposition of linear programs (Dantzig & Wolfe 1960). The Dantzig-Wolfe decomposition guarantees convergence for a specific type of decomposition, but the relationship between abstract actions and SMDPs permits a much more general approach to decomposition. This insight arises from the relationship between the generation of a new abstract actions and asynchronous policy iteration. Consider the two main steps involved in the use of abstract actions:

1. Determine the optimal policy for some region G, under the assumption that the surrounding states have a constant value.
2. Propagate the effects of this policy by keeping the policy in G fixed and propagating values across the states of G.

The first step corresponds to an infinite number of applications of both types of API operators over the region G, while the second type corresponds to an infinite number of value propagation steps (with no policy updates) over G. Thus, the creation and use of a new abstract actions corresponds directly to a specific sequence of API steps. Fortunately, the infinite number value propagation steps can be avoided by solving directly the system of equations as in Section 4. This observation unifies iterative temporal abstraction with API and ensures the convergence of a broad class of iterative abstraction methods. However, this more general *iterative abstraction* algorithm also is guaranteed to converge:

```
Π = initial set of abstract actions
π̂ ← solution to the reduced SMDP induced by Π
Repeat until π̂ is optimal
  Create new abstract action(s)
    Choose some region G
    Determine optimal policy given V_π̂
    Add this policy to Π
  π̂ ← solution to SMDP induced by Π
```

Theorem 5 *Iterative abstraction will converge to the globally optimal policy if a region selection strategy is used that will eventually visit any region in which the policy is not currently optimal.*

Proof Note that this algorithm is more general than API, which covers only the case where a new policy is created and used immediately, but does not address the effects of using old, potentially suboptimal polices that might be left in Π. However, observe that the value of any state in the system is non-decreasing for each iteration and that if the policy is not optimal for some s, then there must be some enclosing G such that when a new abstract action is defined for G, the value of s will improve.■

In general, the computational benefits of iterative methods are difficult to characterize in a problem-independent manner. The number of iterations required will depend largely on the quality of the initial set of abstract actions. A fortuitous set can lead to a drastic reduction in computational effort. The order in which regions have their policies updated also can affect performance. For example, if the policy in room 2 has been updated in a way that changes the values of the connecting states to room 4, but not the connecting states to room 1, there may be no point in updating the policy for room 1. Parr (1998a) investigates several approaches to the construction and use of abstract actions with the aim of minimizing the number of updates to Π.

5.3 Concurrent methods

A common feature in both reductive and iterative methods is the replacement of a group of states with an abstract action defined over those states. There are times, however, when it may be advantageous to leave the original states in the SMDP and deal with both concrete and abstract actions simultaneously. In the four-room example, this would provide the option of either adopting a fixed policy defined over one of the rooms or adopting a policy that is produced from scratch.

At first, the advantage of concurrent methods may seem unclear: Unless the abstract actions were fortuitously chosen to be optimal, the ultimate solution will never use an abstract action. There are several factors that can make concurrent methods desirable in spite of this obvious problem. First, the optimal solution may not be required. If a satisficing solution is desired, the abstract actions may be adequate. If they are not, the concurrent method can still find the optimal solution. Second, the abstract actions can speed up convergence of iterative methods such as policy iteration and value iteration by quickly providing lower bounds on state values. (Note that the system of linear equations used to construct the abstract action in Section 4 also can be used to determine the value of any state visited under the abstract action as a function of the states in which the action terminates.) Finally, in a reinforcement learning context, the use of abstract actions can reduce flailing by forcing the agent to move across large regions of the state space.

A useful trick that concurrent methods can exploit in RL is the fact that any state transition can be treated as an experience in several different, concurrent learning tasks, (Singh 1992b). For example, an agent executing a policy corresponding to an abstract action is also executing concrete actions along the way. The agent is free to update the Q-values for each state-action pair that occurs, learning Q tables for several different subtasks simultaneously (Sutton, Precup, & Singh 1998).

Concurrent methods are discussed in Precup, Sutton and Singh (1997), where convergence results are established for value iteration and policy iteration in MDPs with the addition of abstract actions. In the framework used here, those results follow as a consequence of the convergence of value iteration and policy iteration for SMDPs. These ideas are applied for reinforcement learning in Precup and Sutton (1998), where policies to achieve subtasks are learned and transformed into abstract actions. A new agent that combines abstract actions and concrete actions is shown to learn much faster than one that uses concrete actions alone. A similar approach is used in McGovern, Sutton and Fagg (1997), where the abstract actions have the form of predefined, closed-loop policies designed to achieve specific objectives.

As with reductive and iterative methods, the benefits of concurrent methods will depend upon the choice of abstract actions. Concurrent methods increase the number of actions available at each state and can increase the cost of each iteration in off-line methods such as policy iteration or value iteration. However, if the abstract actions are well-chosen, they will reduce the number of iterations required. In reinforcement learning a poor choice of abstract actions can slow the rate at which the state space is covered, while a good choice can help the agent reach parts of the space it would take a very long time to discover on a random walk of concrete actions.

6 Discussion

This paper presents a general framework for transforming complex policy descriptions over regions of MDPs into abstract actions for SMDPs. The formal properties of the standard algorithms for SMDPs and of the transformations described here allow strong claims about a variety of temporal abstraction methods as a consequence of more general results about SMDPs. Three different methods of using temporal abstractions are described: Reductive methods solve a reduced SMDP to find the optimal use for a set of predefined policies. A broadly defined class of iterative methods solve a reduced MDP as a subroutine, but can find the globally optimal policy by refining an initial set of abstract actions. Concurrent methods do not reduce the state space, but use a fixed set of abstract actions to improve the speed of convergence to an optimal policy.

The understanding of these categories and their formal properties suggests some interesting directions for future research: Since iterative methods use reductive methods as a subroutine, there should be a way to transform any reductive method into an iterative one. Similarly, any reductive method should have the option of making a transition to concurrency if the optimal reductive policy is not of sufficient quality. In general, the space of iterative methods is not very well explored — only one case of the general iterative abstraction algorithm appears to have been investigated and iterative methods have not been used in a reinforcement learning context. Finally, little is known about the tradeoffs between iterative methods and concurrent methods.

An important and difficult issue is the question of how abstract actions are selected in the first place, since it is clear that the benefit that will be reaped from abstraction will depend upon how well suited the abstractions are to the problem at hand. The performance of reductive and concurrent methods is closely tied to the quality of the provided abstract actions, while the performance of iterative methods is linked to both the initial action set and the scheme for selecting and updating regions of the state space.

In spite of the apparent importance of the initial set of abstraction action, the process of specifying these actions has not been formalized. There seem to be three ways of specifying an initial set of policies:

- guesses about state values
- specification of subtasks
- introduction of partially specified policies/programs

The first two are often used in combination when, for example, a subtask solution is constructed by creating a smaller MDP where states that constitute achievement of the subtask become absorbing states with a fixed, guessed, value. The determination of subtasks and task completion values are fairly domain-dependent and ad-hoc.

The use of policy fragments, particularly HAMs (Parr & Russell 1998), has the advantage that the abstractions can be defined in a programmatic way in terms of the agent's sensors. This provides an agent-centered view of abstraction rather than an external view of abstraction, and allows HAMs to be truly "plug-and-play" in the sense that the same data structure can be used without modification for completely different problems. This should not be interpreted as a reason to abandon subtask specifications, since it does not answer the question of where the policy fragments come from in the first place. HAMs were motivated as a means of leveraging expert knowledge in the design of low level controllers to achieve high-level objectives. However, there is no reason why the HAM language could not be augmented to include heuristic information about the values of states or subtasks, unifying different approaches to abstract action selection.

Ultimately, temporal abstraction is just one part of the solution for managing large MDPs. State space abstraction and value function approximation will also play important roles, and their interaction with temporal abstraction is an important area for further research.

7 Acknowledgment

This work was supported in part by ONR grant no. N00014-97-1-0942 and ARO MURI grant no. DAAH04-96-1-0341. The relationship between the author's previous work on temporal abstractions and SMDPs was first suggested by Sridhar Mahadevan at the 1996 AAAI Fall Symposium.

References

Bellman, R. E. 1957. *Dynamic Programming*. Princeton, New Jersey: Princeton University Press.

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

Boutilier, C., and Dearden, R. 1994. Using abstractions for decison theoretic planning with time constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1016–1022. Seattle, Washington: AAAI Press.

Bradtke, S. J., and Duff, M. O. 1995. Reinforcement learning methods for continuous-time Markov decision problems. In *Advances in Neural Information Processing Systems 7: Proceedings of the 1994 Conference*. Denver, Colorado: MIT Press.

Dantzig, G., and Wolfe, P. 1960. Decomposition principle for dynamic programs. *Operations Research* 8(1):101–111.

Dayan, P., and Hinton, G. E. 1993. Feudal reinforcement learning. In Hanson, S. J.; Cowan, J. D.; and Giles, C. L., eds., *Neural Information Processing Systems 5*, 361–368. San Mateo, California: Morgan Kaufman.

Dean, T., and Givan, R. 1997. Model minimization in Markov decision processes. In *Proceedings of the Fourteenth National Conference on Aritificial Intelligence.*, 106–111. Providence, Rhode Island: MIT Press.

Dean, T., and Lin, S.-H. 1995. Decomposition techniques for planning in stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1121–1127. Montreal, Canada: Morgan Kaufmann.

Dietterich, T. G. 1997. Hierarchical reinforcement learning with the MAXQ value function decomposition. Technical report, Department of Computer Science, Oregon State University, Corvallis, Oregon.

Forestier, J.-P., and Varaiya, P. 1978. Multilayer control of large Markov chains. *IEEE Transactions on Automatic Control* AC-23:298–304.

Hauskrecht, M.; Meuleau, N.; Boutilier, C.; Pack Kaelbling, L.; and Dean, T. 1998. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*. to appear.

Jaakkola, T.; Jordan, M.; and Singh, S. P. 1994. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation* 6(6):1185–1201.

Lin, L.-J. 1993. *Reinforcement Learning for Robots Using Neural Networks*. Ph.D. Dissertation, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Mahadevan, S.; Khaleeli, N.; and Marchalleck, N. 1997. Designing agent controllers using discrete-event Markov models. In *Proceedings of the AAAI 97 Fall Symposium on Model-Directed Autonomous Systems*.

McGovern, A.; Sutton, R. S.; and Fagg, A. H. 1997. Roles of macro-actions in accelerating reinforcement learning. In *1997 Grace Hopper Celebration of Women in Computing*.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference, to appear*.

Parr, R. 1996. Policy based clustering for markov decision processes. In *Proceedings of the AAAI 96 Fall Symposium on Learning Complex Behaviors*.

Parr, R. 1998a. Flexible decomposition agorithms for weakly coupled Markov decision processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*. to appear.

Parr, R. 1998b. *Hierarchical Control and Learning for Markov Decision Processes*. Ph.D. Dissertation, Computer Science Division, University of California, Berkeley, California.

Precup, D., and Sutton, R. S. 1998. Multi-time models for temporally abstract planning. In *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference, to appear*.

Precup, D.; Sutton, R. S.; and Singh, S. P. 1997. Planning with closed-loop macro actions. In *Proceedings of the AAAI Fall Symposium*.

Puterman, M. L. 1994. *Markov Decision Processes*. New York: Wiley.

Singh, S. P., and Gullapalli, V. 1993. Asynchronous modified policy iteration with single-side updates. unpublished manuscript.

Singh, S. P. 1992a. Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Proceedings of the Ninth International Conference on Machine Learning*. Aberdeen: Morgan Kaufmann.

Singh, S. P. 1992b. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning* 8(3):323–340.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1998. Between MDPs and semi-MDPs: Learning, planning, and representing knowledge at multiple temporal scales. *in prep.*

Sutton, R. S. 1995. Temporal abstraction in reinforcement learning. In *Proceedings of the Twelfth International Conference on Machine Learning*, 531–539. Tahoe City, CA: Morgan Kaufmann.

Tate, A. 1977. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, 888–893. Cambridge, Massachusetts: IJCAII.