As appeared in the Proceedings of the Ninth Machine Learning Conference, pp. 406-415. 1992.

Scaling Reinforcement Learning Algorithms by Learning Variable Temporal Resolution Models

Satinder P. Singh
Department of Computer Science
University of Massachusetts
Amherst, MA 01003

Abstract

The close connection between reinforcement learning (RL) algorithms and dynamic programming algorithms has fueled research on RL within the machine learning community. Yet, despite increased theoretical understanding, RL algorithms remain applicable to simple tasks only. In this paper I use the abstract framework afforded by the connection to dynamic programming to discuss the scaling issues faced by RL researchers. I focus on learning agents that have to learn to solve multiple structured RL tasks in the same environment. I propose learning abstract environment models where the abstract actions represent "intentions" of achieving a particular state. Such models are variable temporal resolution models because in different parts of the state space the abstract actions span different number of time steps. The operational definitions of abstract actions can be learned incrementally using repeated experience at solving RL tasks. I prove that under certain conditions solutions to new RL tasks can be found by using simulated experience with abstract actions alone.

1 INTRODUCTION

The close connection between reinforcement learning (RL) algorithms and conventional dynamic programming (DP) algorithms (Watkins 1989; Sutton 1990; Barto et al. 1991; Barto et al. 1990; Werbos 1990) has fueled research on RL within the machine learning community. Yet, despite the consequent increase in theoretical understanding, the inability of RL algorithms to scale well to complex tasks has limited their application to simple tasks (but see Tesauro 1992 for an exception). In this paper I use the general and abstract framework afforded by DP to discuss some of the scaling issues faced by RL researchers. I present a solution to one scaling issue that has been neglected by researchers, but is crucial to making RL applicable to more complex tasks.

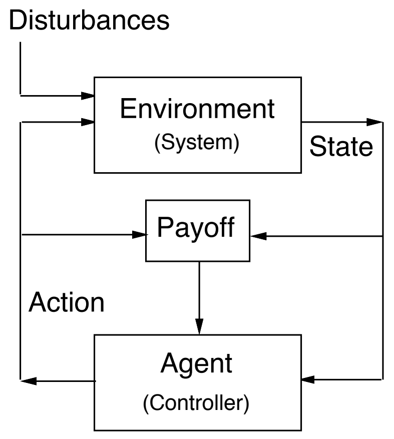

Figure 1: Markovian Decision Task. This figure shows a block diagram representation of a single MDT. It shows an agent interacting with an external environment. The agent observes the state of the environment, executes an action, and gets a payoff in return.

A useful common framework for most RL tasks is obtained by adopting an optimal control perspective. In such a framework an agent interacts with a dynamic external environment and executes actions not only to transform the environment to a desired goal state[1], but also to improve performance with respect to an objective function. A substantial number of applications of RL have dealt with Markovian Decision Tasks[2] (MDTs). MDTs are a subset of discrete-time, optimal

[1]In some RL tasks the objective is not to achieve a desired goal state, but to follow a desired state trajectory optimally, or to simply avoid certain undesirable states. In this paper I do not consider such tasks.

[2]By choosing Markovian decision tasks I ignore the complex issues arising from learning with incomplete state information (e.g., Whitehead et al. 1990), and learning in non-stationary environments.

control tasks with the property that the current state and future actions determine the expected future sequence of states independently of the state trajectory prior to the current state. Figure 1 shows a block diagram representation of an MDT. At each time step the agent observes the state of the environment, executes an action, and receives a payoff in return. In MDTs (Bertsekas 1987) the objective function to be maximized is often of the form, $J_n(i) = \sum_{t=1}^{n} \gamma^t R(t)$, where i is the starting state, $R(t)$ is the expected payoff received at time step t, and n is the horizon of the task. The discount factor, $0 \leq \gamma \leq 1$, determines the weight given to temporally distant payoffs relative to the more immediate payoffs. The horizon determines the time period over which the payoffs are important. A stationary control policy[3], hereafter simply called *policy*, is a function assigning actions to states. A policy that maximizes the agent's objective function is an optimal policy.

For single-stage MDTs, i.e., MDTs that have a horizon of one, the search for an optimal policy can be conducted locally for each state because the optimal action in any state is simply an action that leads to the highest immediate payoff. MDTs with a horizon greater than one, or multi-stage MDTs, face the difficult temporal credit assignment (Sutton 1984) problem. Hence, the search for an optimal action in a state cannot be conducted locally because it may be necessary to examine the consequences of all action sequences of length equal to the horizon of the MDT. If a model of the environment is not known for a multi-stage MDT, the problem faced by the learning agent becomes *non-causal* because constraints on the optimal solution propagate backwards in time from the future. Most algorithms for solving multi-stage MDTs first convert such problems to single-stage MDTs by computing or learning an evaluation function, called a "value" function[4], such that the task of determining an optimal policy given the value function is a single-stage MDT.

Let S be the set of states of the environment, and $\mathcal{A}_1$ be the set of primitive actions (actions executable in one time step in the real environment) available to the agent in each state. Let P denote the state transition probabilities. $P_{xy}(a)$ is the probability of a transition to state y from state x on executing action a. The payoff function, $R(x, a, y)$, is the payoff received by the agent on causing a transition from state x to state y by executing action a. In this paper I focus on the abstract process of learning the value function for infinite-horizon ($n = \infty$), undiscounted ($\gamma = 1$) MDTs, independent of the learning algorithm used. Define the value of state x under policy $\pi : S \rightarrow \mathcal{A}_1$, $V^\pi(x)$, as the infinite horizon sum of the payoff received by the agent if it were to follow the policy π forever from the initial state x. $V^\pi(x)$ can be defined recursively as follows:

$$V^\pi(x) \quad = \quad E\{R(x, \pi(x), y) + V^\pi(y)\},$$

where E is the expectation operator, and y is the next state.

The optimal value of state x, $V^*(x)$, is the value under the optimal policy π^*, and can be defined recursively using the following form of the Bellman optimality equation:

$$V^*(x) \quad = \quad \max_{a \in \mathcal{A}_1} E\{R(x, a, y) + V^*(y)\}.$$

Given the optimal value function the optimal action in state x is determined as follows:

$$\pi^*(x) \quad = \quad \arg\max_{a \in \mathcal{A}_1} E\{R(x, a, y) + V^*(y)\}.$$

Learning the optimal value function is a necessary and almost always a computationally intensive part of learning to solve multi-stage MDTs. I ignore the subsequent process of determining an optimal policy because it does not involve the temporal credit assignment problem. However, it is important to note that determining an optimal policy from the optimal value function can also be computationally intensive, particularly for large action sets. See Gullapalli (1992) for a discussion of the scaling issues involved in deriving the optimal policies for MDTs with large action sets.

2 SCALING ISSUES FOR LEARNING ALGORITHMS BASED ON DYNAMIC PROGRAMMING

Adaptive critic architectures based on Watkins's (1989) Q-learning algorithm, or on Sutton's (1988) temporal differences (TD) algorithm, approximate DP by using repeated experience at actually controlling the environment to incrementally improve their estimate of the optimal value function. Asymptotic convergence results have been obtained under certain conditions for both TD (Sutton 1988) and Q-learning (Watkins 1989; Watkins and Dayan 1992). Sutton (1990) demonstrated that both TD and Q-learning could approximate the optimal value function as well by using simulated experience with a model of the environment. The essential operation shared by all DP-based learning algorithms is that of a "backup". A backup uses a state transition, whether simulated or

[3]Control policies are also referred to as decision policies, or situation-action mappings, or simply as "reactions". I do not consider non-stationary control policies because it is known that the optimal policies for MDTs are stationary (Ross 1983).

[4]The policy iteration algorithm is an exception because it searches for an optimal policy directly in policy space. Nevertheless, policies are evaluated by determining their value function. Therefore the results of this paper are relevant to learning algorithms based on policy iteration.

in the real environment, to update the estimated value of the predecessor state by using the estimated value of the successor state and the immediate payoff for that state transition.

Let $\hat{V}_\pi$ be the estimate of the value function for policy π. The backup equation, assuming knowledge of the state transition probabilities and payoff function, is:

$$\hat{V}_\pi(x) \quad = \quad \sum_{y \in S} P_{xy}(\pi(x))\{R(x, \pi(x), y) + \hat{V}_\pi(y)\} \tag{1}$$

If the transition probabilities are not known, the TD learning rule can be used to update the estimate of the value function as follows:

$$\hat{V}_\pi(x) \quad = (1.0 - \alpha)\hat{V}_\pi(x) + \alpha[R(x, \pi(x), y) + \hat{V}_\pi(y)],$$

where α is the learning rate parameter.

Other DP-based learning (DP-L) algorithms have similar backup equations. I focus on the abstract properties of the backup equation and present results that apply to all DP-L algorithms. Within the above framework, there are two important differences in the DP-L algorithms: the information available to the agent during a backup, and the order in which the backups are performed. DP-L architectures that have access to an accurate environment model can do backups in any arbitrary order, and in addition have potential access to all the information needed for a backup, even for stochastic problems. On the other hand, DP-L algorithms that do not have access to a model, have only sampled information available in stochastic problems, and are limited to backing up into the current state of the environment. For learning tasks where a model of the environment is not available at the beginning, indirect learning algorithms (see Barto and Singh 1990) use system identification techniques to learn a model on-line. Equation 1 can then be used by substituting the estimated transition probabilities for the real transition probabilities.

Numerous researchers have demonstrated accelerated learning in both model-based and model-free approaches by using heuristics and domain knowledge to change the order in which the backups are done. State-preference predicates (Utgoff and Clouse 1991), external critics (Whitehead 1991), external teachers (Lin 1991), and nominal controllers, are some methods of utilizing prior domain knowledge. Smart exploration strategies based on heuristics, such as those used by Sutton (1990), Whitehead et al. (1990), Barto and Singh (1990), Thrun and Moller (1991), and Kaelbling (1990), can also affect the order in which backups are performed by a DP-L algorithm.

While the above heuristic methods do accelerate the process of learning the value function, they have two fundamental limitations: each backup changes the value of only one state (the predecessor state), and each backup involves neighboring states, i.e., states that are linked by primitive actions executable in one time step. The first limitation is addressed in the literature as the state or input generalization issue (Chapman and Kaelbling 1991; Moore 1991). If a function approximator other than a look-up table is used to learn the value function, there will be some generalization across states. However, there has been little research on providing or learning the right generalization bias for learning the value function in arbitrary optimal control tasks (but see Samuel 1967; Yee 1992). One way to achieve perfect generalization for finite-state tasks would be to form state representations that partition the state set into equi-value subsets. Given such a representation, or in the infinite-state case an approximation to one, a single backup can be used to simultaneously update the entire subset of equi-valued states to which the predecessor state belongs.

The much less studied second limitation constitutes the temporal resolution issue. For most control problems there is a finest temporal scale at which the problem can be studied, determined usually by the highest sampling frequency and other hardware constraints. By limiting the backups to that fine a temporal scale, or alternatively to that high a temporal resolution, problems with large state sets become intractable because of the many backups that have to be performed to learn the value function. In this paper I focus exclusively on the temporal resolution issue.

3 VARIABLE TEMPORAL RESOLUTION MODELS

Without a model of the environment a DP-L algorithm has no choice but to do backups at the highest resolution afforded to it in the real environment[5]. To do backups at longer time scales requires an abstract model. Any physical control system can be modeled at any of an infinity of levels of abstraction. The central issue addressed in this paper is the nature of the abstractions appropriate for accelerating learning of the value function for MDTs. In particular, I study the abstractions necessary to mitigate the high temporal resolution problem. To that end, I focus on using abstract models for prescription (see Simon 1990), i.e., on using models to determine the effects of control policies via simulation or temporal projection. However, the models that I will describe could be put to other uses, e.g., for deriving structural explanations to deal with the state generalization issue.

Building abstract models to speed up the process of learning the value function is not a new idea. There is

[5]Alternatively, the controller can decrease the resolution by simply choosing not to change actions at some time steps – but this can only come at the expense of reactivity. Another method of achieving reduced temporal resolution without building abstract models may be Sutton's (1984) method of using eligibility traces to do backups.

some work in doing structural abstractions, i.e., ignoring structural detail about the state that is observed by the agent. I focus on abstracting temporal detail, i.e., the frequency with which the agent observes the state and makes a decision. One way to abstract temporal detail would be to simply learn to make predictions for all possible sequences of actions of a fixed length greater than one. However, the combinatorics of that will outweigh any resulting advantage. Furthermore, it is unlikely that there is a single frequency that will economically capture all that is important to predict. In different parts of the state space of the environment, "interesting" situations, i.e., situations that merit prediction, will occur at different frequencies. Any system identification technique that models the environment at a fixed frequency will be inefficient as compared to a system identification technique that can construct a variable temporal resolution model (VTRM), i.e., a model with different temporal resolutions in different parts of the state space.

I propose learning models for abstract actions that represent the *intention* of achieving interesting situations, and ignore the temporal detail that would have to be taken into account in any operational definition of such abstract actions. Within the above optimal control framework, abstract actions will express intentions of achieving useful environment states. Note that predicting the effect of executing an action, abstract or primitive, requires knowledge of both the state transition probabilities, and the payoff function for that action. While the payoff function for a primitive action is directly available from the environment, the payoff function for an abstract action will clearly depend on the particular control policy that is adopted to realize the intended environment state associated with the abstract action.

3.1 LEARNING TO SOLVE MULTIPLE TASKS

It is unlikely that the computational effort of learning a VTRM would be worthwhile to an agent that has to learn to solve a single MDT. Indeed, for some MDTs it is possible to directly determine an optimal policy by using actual experience at controlling the environment at the highest temporal resolution (Barto and Singh 1990), before the environment model becomes accurate enough to be useful. However, if the learning agent has to learn to solve multiple MDTs (Singh 1992c) in the same environment, the cost of constructing a VTRM can be amortized across the tasks. Figure 2 shows a block diagram representation of multiple MDTs; all the MDTs are defined with the same environment, have the same state set S, the same action set $\mathcal{A}_1$, and the same state transition probabilities P. The payoff function, though, differs for each MDT. For an arbitrary set of MDTs it may be difficult, if not impossible, to determine the useful environment states for forming the abstract actions. The approach I adopt in this paper is to consider learning agents that have to learn to solve a special but useful class of tasks, namely compositionally-structured MDTs, and to use that *a priori* knowledge to help determine the useful abstract actions.

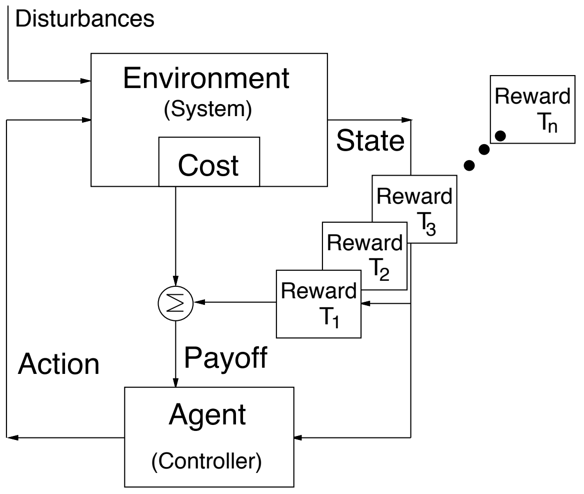

Figure 2: Multiple Markovian Decision Tasks. This figure shows a block diagram representation of multiple MDTs defined in the same environment. The state transition probabilities and the cost function do not change across the tasks; only the reward function does.

Formally, let there be a set of composite MDTs labeled $T_1, T_2, \ldots, T_n$. Each composite MDT requires the agent to learn the optimal path through a sequence of desired states. For example, task $T_i = [x_1 x_2 \cdots x_m]$, where for $1 \leq i \leq m$, $x_i \in S$. Task T_i requires the agent to learn the optimal trajectory from any start state to x_m via *intermediate* states $x_1, x_2, \ldots, x_{m-1}$ in that order. The composite MDTs are compositionally structured because they can be described as a temporal sequence of simpler tasks each of which is an MDT in itself. The task of achieving intermediate state x optimally is itself an MDT, $X = [x]$, defined over the same environment. Without loss of generality, I will assume that the n composite MDTs are defined over a set of N intermediate states labeled $x_1, x_2, \ldots, x_N$. Equivalently, the n composite MDTs are defined over N simpler or elemental MDTs denoted $X_1, X_2, \ldots, X_N$. Note that the symbol X is used to represent an elemental MDT while the symbol T is used to represent a composite MDT. Each MDT is defined via a list of states enclosed in square brackets. The intermediate states of the composite tasks are assumed to be unknown to the learning agent.

The payoff function has two components: $C(x, a)$, the "cost" of executing action a in state x, and $r(x)$, the "reward" for a transition into state x. It is assumed that the cost of executing an action is no greater than zero, and is independent of the task being performed by the learning agent. The reward (Figure 2) for transiting into a state will in general depend on the task. For task T_i, the payoff for executing action a in state x and transiting to state y is $R_i(x,a,y) \doteq r_i(y)+C(x,a)$. To facilitate theoretical analysis, I make the following assumptions:

(A1) Each MDT has a single absorbing goal state. In practice, once the agent reaches the goal state the task is considered to be accomplished and the agent is given the next task. Theoretically, this is equivalent to the agent getting absorbed in that state with zero payoff being provided for every time step after the first time the agent reaches the goal state.

(A2) For all MDTs T_i, $R_i(x,a,y) > 0$ implies that y is the goal state for task T_i and that $x \neq y$.

(A3) Given any state pair, (x,y), there exists a stationary policy that if executed by the agent will transform the environment from initial state x to state y with probability one.

These simplifications and assumptions were introduced for the purpose of theoretical analysis. Even with these simplifications, composite MDTs remain very difficult to solve. Keep in mind that the agent does not know the decomposition of the composite MDTs. In addition, the only reward the agent gets is provided at the very end of the successful completion of a task. Given the long sequences of actions required to solve a composite MDT, the agent faces very difficult temporal credit assignment problems, and conventional RL architectures may be unable to learn the value function for composite MDTs (Singh 1992a). The implications of relaxing some of the above assumptions are discussed in Section 5.

3.2 TEMPORAL ABSTRACTIONS

For compositionally structured tasks, the useful abstract actions would naturally be those that would transform the environment to intermediate states $x_1, x_2, \ldots, x_N$. Thus, abstract action $\mathcal{X}$ represents the "intention" of transforming the environment state to $x \in S$. Figure 3 shows an example of two levels of a hierarchy of VTRMs for a finite-state, deterministic, MDT. The VTRMs are shown as state-transition graphs. The lower graph shows the highest resolution model with the arcs representing primitive actions and the nodes representing states. The upper graph shows two abstract actions $\mathcal{A}$ and $\mathcal{B}$ corresponding to goal states $a, b \in S$. The abstract action $\mathcal{A}$ is shown by direct links from every state to the state marked a. Similarly, for abstract action $\mathcal{B}$ there are direct links to the state marked b from every state. As can be seen by inspecting the abstract model in Figure 3, doing a backup in the abstract model will transmit information between states that are not neighbors in the highest resolution model.

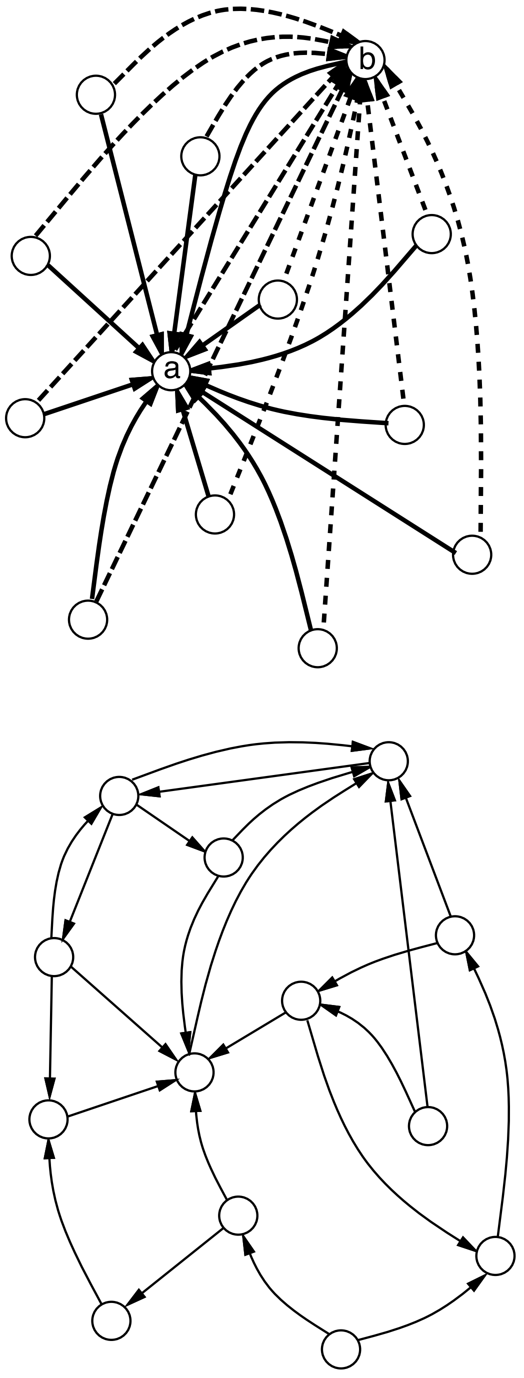

Figure 3: Hierarchy of VTRMs. This figure shows 2 levels of a hierarchy of VTRMs for a deterministic MDT. The lower figure represents the state transition graph with primitive actions as the arcs and the nodes as states. The upper figure shows the same state space with two abstract actions $\mathcal{A}$ and $\mathcal{B}$. The abstract action $\mathcal{A}$ is shown via solid lines and the abstract action $\mathcal{B}$ via dashed lines. The payoffs to be assigned to these arcs will depend on the control policies associated with these abstract actions.

Abstract actions are similar to macro-operators (Korf 1985) in that they allow the agent to ignore irrelevant temporal detail in determining solutions. However, macros are generally open loop sequences of actions that would, if executed, transform the environment from a fixed initial state to a goal state. Macros cannot handle stochasticity and model imperfection because of their open loop nature. The abstract actions I define are closed loop policies for achieving a goal state from any start state and can thus handle stochastic tasks. In addition, since the abstract actions are embedded in an optimal control framework, they can be learned incrementally. Thus, as the policy associated with the abstract action improves, the expected payoff for that abstract action should also get more accurate.

In the next section, I prove that for compositionally structured tasks the abstract actions and their payoffs can be defined in a manner that learning the value function for a new task requires little computation. Thus, temporal abstraction is achieved by learning abstract actions that span many time steps in the real environment.

4 A HIERARCHY OF ENVIRONMENT MODELS

Consider 2 levels of a hierarchy of VTRMs for solving the set of compositionally structured MDTs. Such VTR models are stochastic sequential machines (Booth 1967) of the Mealy-type (when the payoffs are considered to be the outputs of the machines), and when convenient I shall treat them as such. Let M-1 be the highest resolution model or machine with action set $\mathcal{A}_1$ consisting of primitive actions executable in one time step. M-1 has two mappings: the state transition probabilities $P : S \times \mathcal{A}_1 \times S \rightarrow \Re$, and the cost function, $C : S \times \mathcal{A}_1 \rightarrow \Re$. The abstract model M-2 is deterministic, even for a stochastic M-1, and has two mappings: $P_2 : S \times \mathcal{A}_2 \rightarrow S$, and $C_2 : S \times \mathcal{A}_2 \rightarrow R$. $\mathcal{A}_2 = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_N\}$, is the set of abstract actions corresponding to the elemental MDTs $X_1, X_2, \ldots, X_N$ that are defined for the intermediate states $x_1, x_2, \ldots, x_N$. Note that both M-1 and M-2 are mathematical abstractions of the same underlying "real" environment. Machines M-1 and M-2 abstract at different temporal resolutions over the same state set S.

Figure 4 shows that machine M-1 can be used to simulate the abstract machine M-2. Let the mapping $L : \mathcal{A}_2 \times S \rightarrow \mathcal{A}_1$ be defined $\forall \mathcal{X} \in \mathcal{A}_2$, and $\forall s \in S$. Simulating the execution of a single action in M-2 requires following the policy dictated by the mapping L till the first time the state of M-1 equals the state of M-2. Define $F_L(s_0, \mathcal{X}_j)$ to be the expected number of time steps in which the state of M-1 becomes x_j when the simulation for abstract action $\mathcal{X}_j$ is started in state s_0. Let s_i denote the state of M-1 after simulating i steps and let $L(\mathcal{X}_j, s_i) = a_i$. M-2 is said to **realize** the abstract model M-1 under the mapping L, if $\forall s_0 \in S$, and $\forall \mathcal{X}_j \in \mathcal{A}_2$ the following are true: $F_L(s_0, \mathcal{X}_j)$ is bounded from above,

$$\begin{aligned} P_2(s_0, \mathcal{X}_j) &= x_j, \text{ and} \\ C_2(s_0, \mathcal{X}_j) &= E\{\sum_{t=1}^{t=k} R(t)\}, \end{aligned}$$

where $\sum_{t=1}^{t=k} R(t)$ is the k step cumulative payoff received on simulating abstract action $\mathcal{X}_j$ in M-1, starting from state s_0, and k is the number of time steps for M-1 to reach state x_j for the first time.

Define V_i^2 to be the optimal value function for the composite task T_i in the machine defined by M-2. V_i^* is the optimal value function for task T_i. The mapping L is optimal, if $\forall \mathcal{X} \in \mathcal{A}_2$, $\forall s \in S$, and $\forall L'$, $C_{2L}(s, \mathcal{X}) \geq C_{2L'}(s, \mathcal{X})$, where by a slight abuse of notation C_{2L} is the cost function *realized* under mapping L and $C_{2L'}$ is the cost function *realized* under mapping L'. If the agent were to execute the policy defined for abstract action $\mathcal{X}_j$ by an optimal L, it would follow the least cost path to state x_j.

Proposition: If the abstract environment model (M-2) **realizes** the environment model M-1 under an optimal mapping L defined as above, then for all composite tasks T_i, and $\forall s \in S$, $V_i^2(s) = V_i^*(s)$.

Proof: Without loss of generality, consider a composite task $T_i = [x_1 x_2 \ldots x_n]$. By definition, the environment has to go through the states $x_1, x_2, \ldots, x_n$ in sequence. Thus, given the assumptions of the proposition, an optimal policy for task T_i in machine M-2 is to execute the abstract actions in the sequence corresponding to the intermediate states in the decomposition of T_i. For if there were a better policy in M-2, the mapping L would not define optimal policies for all intermediate subtasks. Therefore, for arbitrary start state $s_0 \in S$:

$$\begin{aligned} V_i^2(s_0) &= C_2(s_0, \mathcal{X}_1) + C_2(x_1, \mathcal{X}_2) + \\ &\quad \ldots + C_2(x_{n-1}, \mathcal{X}_n) + r_i(x_n). \end{aligned}$$

Next consider the highest resolution model of the environment, i.e., machine M-1. Let π^* be an optimal policy for composite task T_i. The optimal value of state s_0 for task T_i is given by:

$$V_i^*(s_0) = E_{\pi^*}\{\sum_{t=0}^{t=k_n} C(s_t, a_t^*)\} + r_i(x_n), \quad (2)$$

where a_t^* is an optimal primitive action for state s_t, and k_n is the random variable for the number of steps taken under policy π^* to reach state x_n from start state s_0 via the intermediate states. By definition of T_i, $x_1 x_2 \ldots x_n$ is a subsequence of the set of states visited by the agent under policy π^*. Thus, the sequence of states visited by the agent can be partitioned into

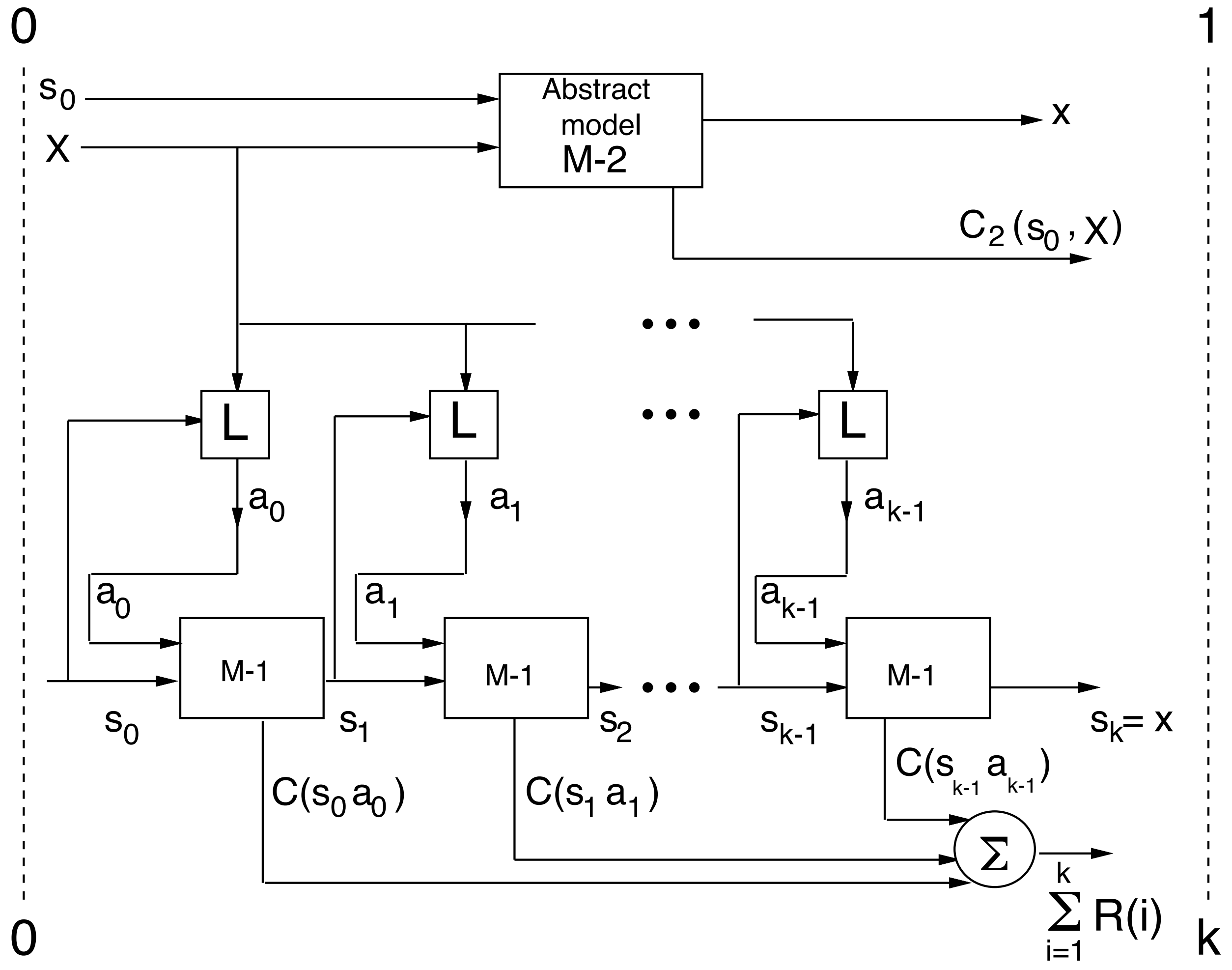

Figure 4: Simulating one step of M-2 in M-1. This figure shows the formal relationship between the abstract environment model (M-2) and the the highest resolution model M-1. The mapping $L : S \times \mathcal{A}_2 \rightarrow \mathcal{A}_1$ is used to simulate M-2 in M-1. The mapping L represents closed loop policies indexed by the abstract actions defined for M-2. Simulating or operationalizing an abstract action requires following the policy dictated by L until the state in M-1 becomes the same as the state in M-2. See text for further details.

subsequences that correspond to the robot getting to the goal locations of the elemental subtasks in the decomposition of T_i. Let π_j^* be an optimal policy for the intermediate subtask $\mathcal{X}_j$. Let k_i be the random variable that represents the time at which state x_i is first visited. Equation 2 can be rewritten as:

$$\begin{aligned} V_i^*(s_0) &= E_{\pi^*}\{\sum_{t=0}^{t=k_1} C(s_t, a_t^*) + \sum_{t=k_1+1}^{t=k_2} C(s_t, a_t^*) + \\ &\quad \ldots + \sum_{t=k_{n-1}+1}^{t=k_n} C(s_t, a_t^*)\} + r_i(x_n). \quad (3) \end{aligned}$$

Solving the composite task T_i optimally would require each of the subtasks to be solved optimally (by definition of composite tasks and the definition of optimality). Once the agent reaches the final goal state x_n the remaining payoffs are zero, and hence Equation 3 can be rewritten as:

$$\begin{aligned} V_i^*(s_0) &= E_{\pi_1^*}\{\sum_{t=0}^{t=k_1} C(s_t, a_t)\} + \\ &\quad E_{\pi_2^*}\{\sum_{t=k_1+1}^{t=k_2} C(s_t, a_t)\} + \ldots \\ &\quad + E_{\pi_n^*}\{\sum_{t=k_{n-1}+1}^{t=k_n} C(s_t, a_t)\} + r_i(x_n). \end{aligned}$$

By assumption, $C(s_t, a_t)$ is independent of the task being performed $\forall s_t \in S$, and $a_t \in \mathcal{A}_1$. Therefore, if M-1 *realizes* M-2 and the policies defined under L are optimal:

$$\begin{aligned} V_i^*(s_0) &= C_2(s_0, \mathcal{X}_1) + C_2(x_1, \mathcal{X}_2) + \ldots \\ &\quad + C_2(x_{n-1}, \mathcal{X}_n) + r_i(x_n) \end{aligned}$$

Therefore for all composite tasks T_i, and $\forall s \in S; V_i^2(s) = V_i^*(s)$.

Q.E.D.

The proposition proved above implies that after building M-2, which requires learning to solve the elemental MDTs, composite MDTs can be solved by doing dynamic programming only in M-2. Value functions for composite MDTs can be learned faster by doing backups in the abstract model M-2 because information about the value function gets propagated backwards from the goal further in one backup than it would with a single backup in M-1.

5 DISCUSSION

The choice of compositional structure for the set of MDTs made the problem of determining the useful temporal abstractions relatively straightforward. The important and difficult question of the automatic discovery of useful abstract actions for classes of tasks more general than compositionally structured tasks is not addressed here. However, the idea that abstract actions should represent intentions of transforming the environment to useful, "landmark" states could generalize to an arbitrary set of tasks. For example, for an animal in the real world there are many landmark states, locations of food, water, shelter, etc., that could serve as goal states for abstract actions. Learning to solve a new task could then simply plan in terms of realizing the intention of being in the "food" state without regard to the temporal detail of realizing the intention.

Another special aspect of compositionally-structured tasks is that the optimal solutions for composite tasks could be constructed by temporally concatenating the optimal solutions to the appropriate elemental tasks. While that may not be true for an arbitrary set of tasks, it may still be possible to construct near-optimal solutions to complex tasks using the solutions to simpler subtasks. Doing DP in abstract models may then *quickly* lead to a near-optimal value function. The quick sub-optimal solutions could then be optimized over time using experience in the real environment. Further research is needed to test these intuitions.

Learning abstract actions shares some drawbacks with macro-learning systems (e.g., Iba 1989); they increase the total size of the action set available to the agent (increased branching factor), and redundant and useless abstract actions can offset the advantages gained from reducing the potential number of backups. It is possible that many of the heuristics used in macro-learning systems to provide partial remedies to the above problems may transfer to learning abstract actions.

The purpose of this paper was to prove that given the correct temporal abstractions for the special class of tasks under study here, the value function for a new composite task could be learned by doing backups from simulated experience in the abstract model alone. To that end, I assumed that the desired abstract actions had already been learned in M-2. In a separate paper (Singh 1992b), I address the issue of *learning* the useful abstract actions for a set of compositionally-structured MDTs. Note that the learning agent does not have access to the structure of the composite tasks, else the discovery would be trivial. In Singh (1992b), I also present a learning architecture that learns to solve a set of compositionally-structured MDTs using an extension to Sutton's (1990) DYNA architecture. On simulations performed on a robot navigation task, significant savings were achieved using abstract models over a learning system that uses the same learning algorithm, but learns with the highest resolution model only.

6 CONCLUSION

Machine learning researchers, whether they study supervised learning tasks, single-stage RL tasks, or multi-stage RL tasks have to abstract structural detail in order to generalize to unseen parts of the input space. Of the three types of learning tasks mentioned above, the need to abstract temporal detail arises only for researchers studying multi-stage RL tasks (but see Schmidhuber 1992). One of the contributions of this paper is to demonstrate that temporal detail can be abstracted independently of abstracting structural detail, and that the two are really orthogonal issues. Of course, both these issues will have to be resolved simultaneously before RL algorithms can deal with most "real" optimal control tasks. The insight that they can be tackled independently could lead to new techniques for addressing them jointly.

Most applications of RL algorithms have been to single tasks. An additional contribution of this paper is to emphasize the need to study autonomous learning agents that have to learn to solve multiple tasks. There are at least three reasons to do so: building realistic autonomous robots will require that ability, research on achieving transfer of learning across multiple tasks is an important approach to the scaling problem, and it could lead to insights for handling non-stationary environments.

Acknowledgments

I thank Andy Barto and Rich Sutton for many inspiring and insightful conversations, as well as for their extensive guidance and help in formulating my ideas. I also thank Andy Barto, Rich Sutton, Richard Yee, Steve Bradtke, and Vijaykumar Gullapalli for providing valuable comments on early drafts. This material is based upon work supported by the Air Force Office of Scientific Research, Bolling AFB, under Grant AFOSR-89-0526 and by the National Science Foundation under Grant ECS-8912623.

References

Barto, A. G., Bradtke, S., & Singh, S. P. (1993). Learning to act using real-time dynamic programming. Technical Report 93-02, University of Massachusetts, Amherst, MA. Submitted to: AI Journal.

Barto, A. G. & Singh, S. P. (Nov. 1990). On the computational economics of reinforcement learning. In *Proceedings of the 1990 Connectionist Models Summer School*, San Mateo, CA. Morgan Kaufmann.

Barto, A. G., Sutton, R. S., & Watkins, C. (1990). Sequential decision problems and neural networks. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems 2*, pages 686–693, San Mateo, CA. Morgan Kaufmann.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models.* Englewood Cliffs, NJ: Prentice-Hall.

Booth, T. L. (1967). *Sequential Machines and Automata Theory.* John Wiley.

Chapman, D. & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the 1991 International Joint Conference on Artificial Intelligence.*

Gullapalli, V. (1992). *Reinforcement Learning and its application to control.* PhD thesis, University of Massachusetts, Amherst, MA 01003.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning, 3*, 285–317.

Kaelbling, L. P. (1990). *Learning in Embedded Systems.* PhD thesis, Stanford University, Department of Computer Science, Stanford, CA. Technical Report TR-90-04.

Korf, R. E. (1985). *Learning to Solve Problems by Searching for Macro-Operators.* Massachusetts: Pitman Publishers.

Lin, L. J. (1991). Self-improvement based on reinforcement learning, planning and teaching. In Birnbaum, L. & Collins, G. (Eds.), *Maching Learning: Proceedings of the Eighth International Workshop*, pages 323–327, San Mateo, CA. Morgan Kaufmann.

Moore, A. W. (1991). Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In Birnbaum, L. A. & Collins, G. C. (Eds.), *Maching Learning: Proceedings of the Eighth International Workshop*, pages 333–337, San Mateo, CA. Morgan Kaufmann.

Ross, S. (1983). *Introduction to Stochastic Dynamic Programming.* New York: Academic Press.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal on Research and Development*, 601–617.

Schmidhuber, J. H. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation, 4*, 234–243.

Simon, H. A. (1990). Prediction and prescription in systems modeling. *Operations Research, 38*(1), 7–14.

Singh, S. P. (1992a). The efficient learning of multiple task sequences. In Moody, J. E., Hanson, S. J., & Lippman, R. P. (Eds.), *Advances in Neural Information Processing Systems 4*, pages 251–258, San Mateo, CA. Morgan Kauffman.

Singh, S. P. (1992b). Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 202–207, San Jose,CA. AAAI Press/MIT Press.

Singh, S. P. (1992c). Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning*, *8*(3/4), 323–339.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S. (1990). Integrating architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the Seventh International Conference on Machine Learning*, pages 216–224, San Mateo, CA. Morgan Kaufmann.

Tesauro, G. J. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*(3/4), 257–277.

Thrun, S. B. & Möller, K. (1992). Active exploration in dynamic environments. In Moody, J. E., Hanson, S. J., & Lippman, R. P. (Eds.), *Advances in Neural Information Processing Systems 4*, San Mateo, CA. Morgan Kauffman.

Utgoff, P. E. & Clouse, J. A. (1991). Two kinds of training information for evaluation function learning. In *Proceedings of the Ninth Annual Conference on Artificial Intelligence*, pages 596–600, San Mateo, CA. Morgan Kaufmann.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis, Cambridge Univ., Cambridge, England.

Watkins, C. J. C. H. & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3/4), 279–292.

Werbos, P. J. (1990). Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, *3*(2), 179–189.

Whitehead, S. D. (1991). Complexity and cooperation in Q-learning. In Birnbaum, L. & Collins, G. (Eds.), *Maching Learning: Proceedings of the Eighth International Workshop*, pages 363–367, San Mateo, CA. Morgan Kaufmann.

Whitehead, S. D. & Ballard, D. H. (1990). Active perception and reinforcement learning. In *Proc. of the Seventh International Conference on Machine Learning*, Austin, TX. M.

Yee, R. C. (1992). Abstraction in control learning. Technical Report COINS Technical Report 92-16, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003. A dissertation proposal.