Spatiotemporal Abstraction of Stochastic Sequential Processes

Sridhar Mahadevan

Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003, USA
mahadeva@cs.umass.edu

Abstract. Probabilistic finite state machines have become a popular modeling tool for representing *sequential processes*, ranging from images and speech signals to text documents and spatial and genomic maps. In this paper, I describe two hierarchical abstraction mechanisms for simplifying the (estimation) learning and (control) optimization of complex Markov processes: *spatial* decomposition and *temporal* aggregation. I present several approaches to combining spatial and temporal abstraction, drawing upon recent work of my group as well as that of others. I show how spatiotemporal abstraction enables improved solutions to three difficult sequential estimation and decision problems: hidden state modeling and control, learning parallel plans, and coordinating with multiple agents.

1 Introduction

Abstraction has long been viewed as central to artificial intelligence (AI). A popular textbook defines abstraction as the "process of removing detail from a representation" [30]. Many approaches to abstraction have been pursued in the past several decades of research in AI. A common strategy is *constraint relaxation* where the problem is simplified by eliminating some conditions, as illustrated by logic-based planners such as ABSTRIPS [12] and by methods for discovering admissible heuristics [25]. This paper describes some recent work on *probabilistic abstraction* of stochastic sequential processes, which have become a common approach underlying many areas of AI.

Figure 1 characterizes a popular view of AI as the science underlying the design of *agents*: software or hardware artifacts that interact with an external environment through perception and action. What is unique about the agent-centered viewpoint is that it directs attention to the *sequential* interaction between an agent and its environment, and how to model the dynamics of such an interaction. Typically, the interaction is such that decisions (or observations) made earlier can impact later decisions.

Probabilistic finite state machines have become a popular paradigm for modeling sequential processes. In this representation, the interaction between an

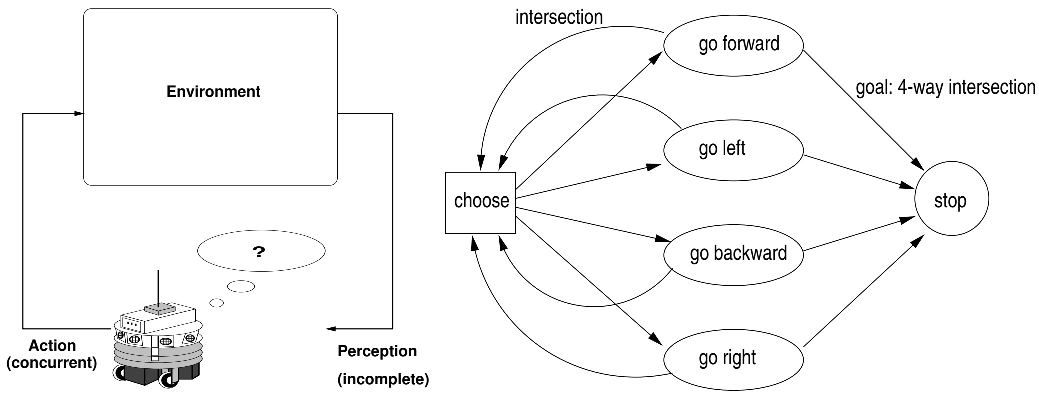

Fig. 1. The perception-action cycle of interaction between an agent and its environment can be modeled as a sequential process. A sequential program for a corridor navigation task represented as a finite state machine on the right. The set of observations generated as the agent executes this machine can be modeled as a Markov process.

agent and its environment is represented as a finite automata, whose *states* partition the past history of the interaction into equivalence classes, and whose *actions* cause (probabilistic) transitions between states. Here, state are a *sufficient statistic* for computing optimal (or best) actions, meaning past history leading to the state can be abstracted. This assumption is usually referred to as the *Markov* property.

Markov processes have become the mathematical foundation for much current work in reinforcement learning [33], decision-theoretic planning [1], information retrieval [7], speech recognition [10], active vision [20], and robot navigation [13]. In this paper, I focus on the abstraction of sequential Markov processes, and present two main strategies for "removing irrelevant detail": state aggregation/decomposition and temporal abstraction. State decomposition methods typically represent states as collections of *factored* variables [1], or simplify the automaton by eliminating "useless" states [3]. Temporal abstraction mechanisms, for example in hierarchical reinforcement learning [34, 5, 23], encapsulate lower-level observation or action sequences into a single unit at more abstract levels. For a unified algebraic treatment of abstraction of Markov decision processes that covers both spatial and temporal abstraction, the reader is referred to the paper by Ravi and Barto in these proceedings [27].

The main thesis of this paper is that combining spatial and temporal abstraction enables significant advances in solving difficult sequential estimation and decision problems. I focus on three specific problems – multiscale representations of hidden state, learning concurrent plans, and acquiring multiagent coordination strategies – and show how spatiotemporal abstraction is a powerful approach to solving instances of these well-known difficult problems.

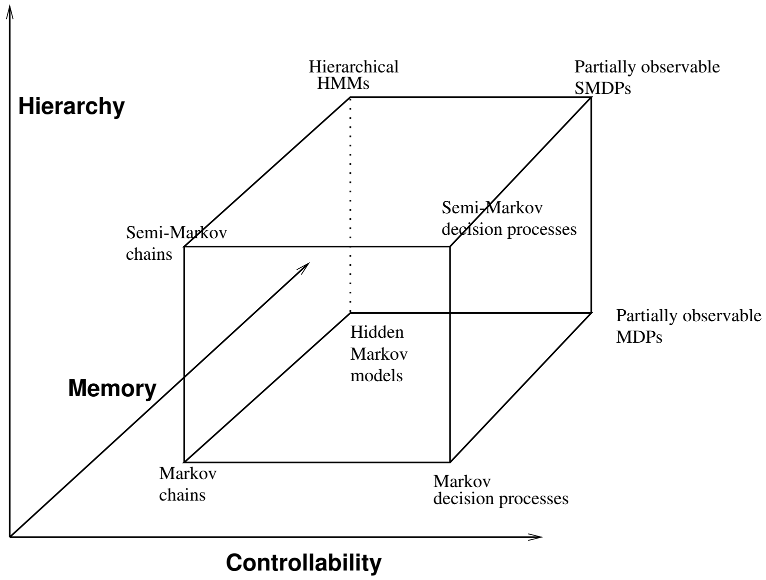

Fig. 2. A spectrum of Markov process models along several dimensions: whether agents have a choice of action, whether states are observable or hidden, and whether actions are unit-time (single-step) or time-varying (multi-step).

2 Markov Processes

Figure 2 illustrates eight Markov process models, arranged in a cube whose axes represent significant dimensions along which the models differ from each other. While a detailed description of each model is beyond the scope of this paper, I will provide examples of many of these models in the rest of this paper, beginning in this section with the basic MDP model.

A *Markov decision process* (MDP) [26] is specified by a set of states S, a set of allowable actions $A(s)$ in each state s, and a transition function specifying the next-state distribution $P^a_{ss'}$ for each action $a \in A(s)$. A reward or cost function $r(s,a)$ specifies the *expected* reward for carrying out action a in state s. Solving a given MDP requires finding an optimal mapping or *policy* $\pi^* : S \rightarrow A$ that maximizes the long-term cumulative sum of rewards (usually discounted by some factor $\gamma < 1$) or the expected average-reward per step. A classic result is that for any MDP, there exists a stationary deterministic optimal policy, which can be found by solving a nonlinear set of equations, one for each state (such as by a successive approximation method called *value iteration*):

$$V^*(s) = \max_{a \in A(s)} \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right) \tag{1}$$

MDPs have been applied to many real-world domains, ranging from robotics [13, 16] to engineering optimization [2, 17, 17], and game playing [36]. In many such domains, the model parameters (rewards, transition probabilities) are unknown, and need to be estimated from samples generated by the agent exploring the environment. Q-learning was a major advance in direct policy learning, since it obviates the need for model estimation [41]. Here, the Bellman optimality equation is reformulated using *action values* $Q^*(x, a)$, which represent the value of the non-stationary policy of doing action a once, and thereafter acting optimally. Q-learning eventually finds the optimal policy asymptotically. However, much work is required in scaling Q-learning to large problems, and abstraction is one of the key components. Factored approaches to representing value functions are also emerging as a key approach to scaling to large problems [14].

3 Spatiotemporal Abstraction of Markov Processes

We now discuss strategies for hierarchical abstraction of Markov processes, including temporal abstraction, and spatial abstraction techniques.

3.1 Semi-Markov Decision Processes

Hierarchical decision-making models require the ability to represent lower-level policies over primitive actions as primitive actions at the next level (e.g., in the sequential machine in Figure 1, the "go forward" state might itself be comprised of a lower-level machine for moving through the corridor to the end, while avoiding obstacles). Policies over primitive actions are "semi-Markov" at the next level up, and cannot be simply treated as single-step actions over a coarser time scale over the same states.

Semi-Markov decision processes (SMDPs) have become the preferred language for modeling temporally extended actions. Unlike Markov decision processes (MDPs), the time between transitions may be several time units and can depend on the transition that is made. An SMDP is defined as a five tuple (S,A,P,R,F), where S is a finite set of states, A is the set of actions, P is the state and action transition probability function, R is the reward function, and F is a function giving probability of transition times for each state-action pair. The transitions are at decision epochs only. The SMDP represents snapshots of the system at decision points, whereas the so-called *natural process* [26] describes the evolution of the system over all times. Discrete-time SMDPs represent transition distributions as $F(s', N|s, a)$, which specifies the expected number of steps N that action a will take before terminating in state s' starting in state s. For continuous-time SMDPs, $F(t|s, a)$ is the probability that the next decision epoch occurs within t time units after the agent chooses action a in state s at a decision epoch. Q-learning generalizes nicely to discrete and continuous-time SMDPs. The Q-learning rule for discrete-time discounted SMDPs is

$$Q_{t+1}(s, a) \leftarrow Q_t(s, a)(1 - \beta) + \beta \left(R + \gamma^k \max_{a' \in A(s')} Q_t(s', a') \right)$$

where $\beta \in (0, 1)$, and action a was initiated in state s, lasted for k steps, and terminated in state s', while generating a total discounted sum of rewards of R.

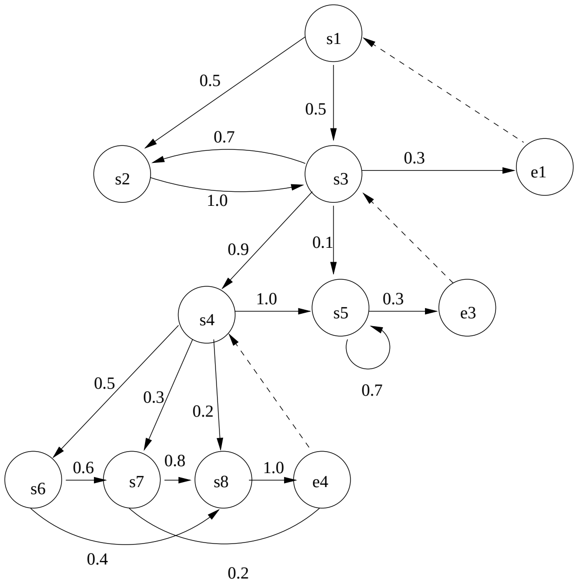

Fig. 3. An example hierarchical hidden Markov model. Only leaf nodes produce observations. Internal nodes can be viewed as generating sequences of observations.

Several frameworks for hierarchical reinforcement learning have been proposed, all of which are variants of SMDPs, including options [34], MAXQ [5], and HAMs [23]. We discuss some of these in more detail in the next section.

3.2 Hierarchical Hidden Markov Models

Hidden Markov models (HMMs) are a widely-used probabilistic model for representing time-series data, such as speech [10]. Unlike an MDP, states are not perceivable, and instead the agent receives an observation o which can be viewed as being generated by a stochastic process $P(o|s)$. HMMs have been widely applied to many time-series problems, ranging from speech recognition [10], information extraction [7], and bioinformatics [11]. However, like MDPs, HMMs do not provide any direct way of representing higher-level structure that is often present in many practical problems. For example, an HMM can be used as a spatial representation of indoor enviroments [31], but typically such environments have higher order structures such as corridors or floors which are not made explicit in the underlying HMM model. As in the case with MDPs, in most practical problems, the parameters of the underlying HMM have to be learned from samples. The most popular method for learning an HMM model is the Baum-Welch procedure, which is itself a special case of the more general Expectation-Maximization (EM) statistical inference algorithm.

Recently, an elegant hierarchical extension of HMMs was proposed [6]. The HHMM generalizes the standard hidden Markov model by allowing hidden states to represent stochastic processes themselves. An HHMM is visualized as a tree structure (see Figure 3) in which there are three types of states, production states (leaves of the tree) which emit observations, and internal states which are (unobservable) hidden states that represent entire stochastic processes. Each production state is associated with an observation vector which maintains distribution functions for each observation defined for the model. Each internal state is associated with a horizontal transition matrix, and a vertical transition vector. The horizontal transition matrix of an internal state defines the transition probabilities among its children. The vertical transition vectors define the probability of an internal state to activate any of its children. Each internal state is also associated with a child called an *end-state* which returns control to its parent. The end-states ($e1$ to $e4$ in Figure 3) do not produce observations and cannot be activated through a vertical transition from their parent.

Figure 3 shows a graphical representation of an example HHMM. The HHMM produces observations as follows:

1. If the current node is the root, then it chooses to activate one of its children according to the vertical transition vector from the root to its children.
2. If the child activated is a product state, it produces an observation according to an observation probability output vector. It then transitions to another state within the same level. If the state reached after the transition is the end-state, then control is returned to the parent of the end-state.
3. If the child is an abstract state then it chooses to activate one of its children. The abstract state waits until control is returned to it from its child end-state. Then it transitions to another state within the same level. If the resulting transition is to the end-state then control is returned to the parent of the abstract state.

The basic inference algorithm for hierarchical HMMs is a modification of the "inside-outside" algorithm for stochastic context-free grammars, and runs in $O(T^3)$ where T is the length of the observation sequence. Recently, Murphy developed a much faster inference algorithm for hierarchical HMMs by mapping them onto a dynamic Bayes network [21].

3.3 Factorial Markov Processes

In many domains, states are comprised of collections of objects, each of which can be modeled as a multinomial or real-valued variable. For example, in driving, the state of the car might include the position of the accelerator and brake, the radio, the wheel angle etc. Here, we assume the agent-environment interaction can be modeled as a factored semi-Markov decision process, in which the state space is spanned by the Cartesian product of random variables $X = \{X_1, X_2, ..., X_n\}$, where each X_i takes on values in some finite domain $Dom(X_i)$. Each action is either a primitive (single-step) action or a closed-loop policy over primitive actions.

Dynamic Bayes networks (DBNs) [4] are a popular tool for modeling transitions across factored MDPs. Let X_i^t denote the state variable X_i at time t

and X_i^{t+1} the variable at time $t+1$. Also, let A denote the set of underlying primitive actions. Then, for any set of actions represented by $\boldsymbol{a} \subseteq A$, the *Action Network* is specified as a two-layer directed acyclic graph whose nodes are $\{X_1^t, X_2^t, ..., X_n^t, X_1^{t+1}, X_2^{t+1}, ..., X_n^{t+1}\}$ and each node X_i^{t+1} is associated with a *conditional probability table (CPT)* $P(X_i^{t+1}|\phi(X_i^{t+1}), \boldsymbol{a})$ in which $\phi(X_i^{t+1})$ denotes the parents of X_i^{t+1} in the graph. The transition probability $P(X^{t+1}|X^t, \boldsymbol{a})$ is then defined by: $P(X^{t+1}|X^t, \boldsymbol{a}) = \prod_i^n P(X_i^{t+1}|w_i, \boldsymbol{a})$ where w_i is a vector whose elements are the values of the $X_j^t \in \phi(X_i^{t+1})$.

Figure 4 shows a popular toy problem called the Taxi Problem [5]) in which a taxi inhabits a 7-by-7 grid world. This is an episodic problem in which the taxi (with maximum fuel capacity of 18 units) is placed at the beginning of each episode in a randomly selected location with a randomly selected amount of fuel (ranging from 8 to 15 units). A passenger arrives randomly in one of the four locations marked as R(ed), G(reen), B(lue), and Y(ellow) and will select a random destination from these four states to be transported to. The taxi must go to the location of the passenger (the "source"), pick up the passenger, move to the destination location (the "destination) and put down the passenger there. The episode ends when either the passenger is transported to the desired destination, or the taxi runs out of fuel. Treating each of taxi position, passenger location, destination and fuel level as state variables, we can represent this problem as a factorial MDP with four state variables each taking on values as explained above. Figure 4 shows a factorial representation of taxi domain for *Pickup* and *Fillup* actions.

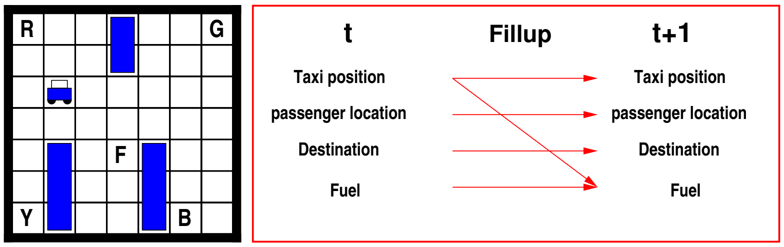

Fig. 4. The taxi domain is an instance of a factored Markov process, where actions such as fillup can be represented compactly using dynamic Bayes networks.

Other examples include the *mixed memory factorial Markov model* and its extension to factorial MDPs [29]. In factorial MDPs, the transition model is additionally decomposed into transition sub-components, each describing a transition sub-model for a particular state variable given the current instantiation of the set of state variables. Mixed memory representation of the transition probabilities models each sub-component as a mixture of simpler dynamical models, each describing a cross-correlation between a pair of state variables. The parameters of this model can be fitted iteratively using an EM procedure.

3.4 Structural Decomposition of Markov Processes

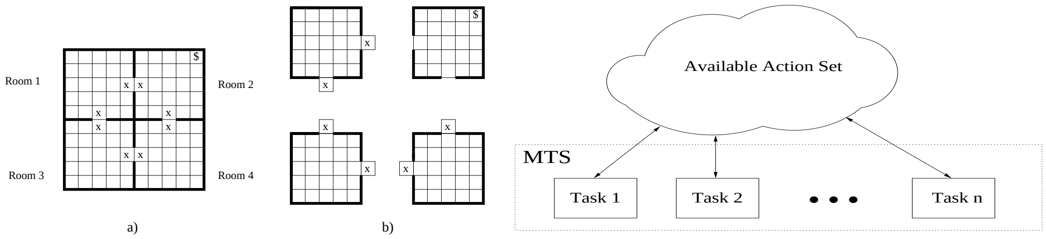

Fig. 5. State and action-based decomposition of Markov processes.

Other related techniques for decomposition of large MDPs have been explored, and some of these are illustrated in Figure 5. A simple decomposition strategy is to split a large MDP into sub-MDPs, which interact "weakly" [23, 34, 3]. An example of weak interaction is navigation, where the only interaction among sub-MDPs is the states that connect different rooms together. Another strategy is to decompose a large MDP using the set of available actions, such as in air campaign planning problem [19], or in conversational robotics [24]. An even more intriguing decomposition strategy is when sub-MDPs interact with each other through shared parameters. The transfer line optimization problem from manufacturing is a good example of such a parametric decomposition [40].

4 Spatiotemporal Abstraction: Three Case Studies

This section describes some recent research from my group on exploiting spatiotemporal abstraction to produce improved solutions to three difficult problems: learning concurrent actions and multiagent coordination, and using memory to deal with hidden state.

4.1 Learning Concurrent Plans

I now describe a probabilistic model for learning concurrent plans over temporally extended actions. Figure 6 illustrates a toy example of concurrent planning. The general problem is as follows. The agent is given a set of temporally extended actions, each of which can be viewed as a (fixed or previously learned) "subroutine" for choosing actions over a subspace of the overall state space. The goal of the agent is to learn to construct a closed-loop plan (or policy) that allows multiple concurrent subroutines to be executed in parallel (and in sequence) to achieve the task at hand. For multiple actions to be executed concurrently, their joint semantics must be well-defined. Concurrency is facilitated by assuming states are not atomic, but structured as a collection of (discrete or continuous)

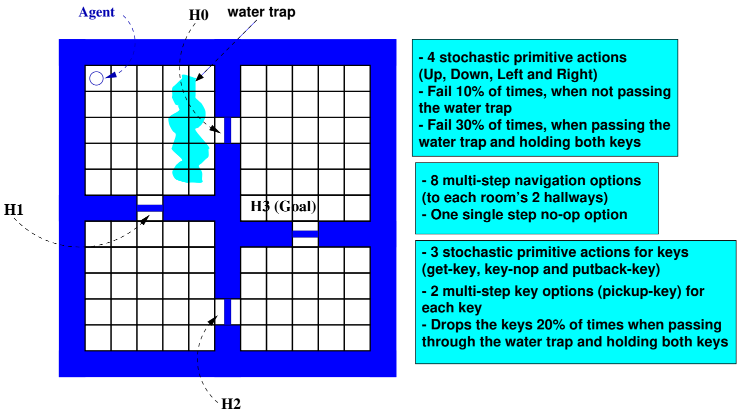

Fig. 6. A grid world problem to illustrate concurrent planning: the agent is given subroutines for getting to each door from any interior room state, and for opening a locked door. It has to learn the shortest path to the goal by concurrently combining these subroutines. The agent can reach the goal more quickly if it learns to parallelize the subroutine for retrieving the key before it reaches a locked door. However, retrieving the key too early is counterproductive since it can drop with some probability.

variables, and the effect of actions on such sets of variables can be captured by a compact representation, such as a dynamic Bayes net (DBN) [4].

Since multiple concurrent actions may not terminate synchronously, the notion of a decision epoch needs to be generalized. For example, a decision epoch can occur when any one of the actions currently running terminates. We refer to this as the T_{any} termination condition. Alternatively, a decision epoch can be defined to occur when all actions currently running terminate, which we refer to as the T_{all} condition.

For concreteness, we will describe the concurrent planning framework using the *options* formalism [34]. The treatment here is restricted to options over discrete-time SMDPs and deterministic policies, but the main ideas extend readily to other hierarchical formalisms [5, 23] and to continuous-time SMDPs [8]. More formally, an option o consists of three components: a policy $\pi : S \to A$, a termination condition $\beta : S \to [0,1]$, and an initiation set $I \subseteq S$, where I denotes the set of states s in which the option can be initiated. For any state s, if option π is taken, then primitive actions are selected based on π until it terminates according to β. An option o is a *Markov option* if its policy, initiation set and termination condition depend stochastically only on the current state $s \in S$. An option o is *semi-Markov* if its policy, initiation set and termination condition are dependent on all prior history since the option was initiated. For example, the option *exit-room* in the grid world environment shown in Figure 6, in which states are the different locations in the room, is a Markov option, since for a given location, the direction to move to get to the door can be computed given the current state.

A hierarchical policy over subroutines or options can be defined as follows. The Markov policy over options $\mu : S \rightarrow O$ (where O is the set of all options) selects an option $o \in O$ at time t using the function $\mu(s_t)$. The option o is then initiated in s_t until it terminates at a random time $t + k$ in some state s_{t+k} according to the termination condition, and the process repeats in s_{t+k}.

The multistep state transition dynamics over options is defined using the discount factor to weight the probability of transitioning. Let $p^o(s, s', k)$ denote the probability that the option o is initiated in state s and terminates in state s' after k steps. Then $p(s'|s, o) = \sum_{k=1}^{\infty} p^o(s, s', k)\gamma^k$. If multi-step models of options and rewards are known, optimal hierarchical plans can be found by solving a generalized Bellman equation over options similar to Equation 1.

The sequential subroutine (option) model is now generalized to *concurrent multi-options*. Initially, assume the set of options can be partitioned into mutually exclusive *coherent* subsets, where options from disjoint subsets can always be parallelized since they affect different state variables. For example, turning the radio off and pressing the brake can always be executed in parallel since they affect different state variables.

We now define multi-options (denoted below by $\boldsymbol{o}$) more precisely. Let $o \equiv < I, \pi, \beta >$ be a standard option with state space S_o governed by the set of state variables $W_o = \{w_1^o, w_2^o, ..., w_{n_o}^o\}$. Let $\varphi_o \subset W_o$ denote the subset of state variables that evolve by some other processes (e.g. other options) and independent of o, and let $\Omega_o = W_o - \varphi_o$ denote the subset of state variables that evolve solely based on the option o. We refer to the class of options with this property as *partially-factored* options. Two options o_i and o_j are called *coherent* if (1) they are both *partially-factored* options, and (2) $\Omega_{o_i} \cap \Omega_{o_j} = \emptyset$ (this condition ensures these two options will not affect the same portion of the state space so that they can safely run in parallel). In the driving example, *turn radio on* and *brake* options are coherent, but *turn-right* and *accelerate* options are not coherent, since the state variable *position* is controlled by both these actions.

Now, assume O is the set of available options and $\{C_1, C_2, ..., C_n\}$ are n classes of options that partition O into disjoint classes such that any two options belonging to different classes are coherent (can run in parallel), and any two options within the same class are not coherent. Clearly any set of options generated by drawing each option from a separate class can safely be run in parallel. Given the above definitions, we can define the multi-option model as a 4-tuple $(S, \boldsymbol{O}, P, r)$, where S is the state space spanned by the cartesian product set over state variables, $\boldsymbol{O}$ is the set of all possible concurrent multi-options $\subseteq C_1 \times C_2 \times ... \times C_n$, P is the transition probability specifying the state dynamics under any multi-option, and $r : S \times \boldsymbol{O} \rightarrow \mathcal{R}$ is the expected reward for taking a multi-option $\boldsymbol{o} \in \boldsymbol{O}$.

When multi-option $\boldsymbol{o}$ is executed in state s, a set of m options $o_i \in \boldsymbol{o}$ are initiated. Each option o_i will terminate at some random time t_{o_i}. We can define the event of *termination* for a multi-option based on either of the following events:

(1) $T_{all} = \max_i(t_{o_i})$: when all the options $o_i \in \boldsymbol{o}$ terminate according to $\beta_i(s)$, multi-option $\boldsymbol{o}$ is declared terminated (2) $T_{any} = \min_i(t_{o_i})$: when any (i.e., the first) of the options terminate, the rest of the options that are not terminated at that point in time are interrupted. Under either definition of termination, the following result holds.

Theorem 1. *Given a Markov decision process, and a set of concurrent Markov options defined on it, the decision process that selects only among multi-options, and executes each one until its termination according to the either T_{all} or T_{any} termination condition forms a semi-Markov decision process.*

The proof requires showing that the state transition dynamics $p(s', N \mid \boldsymbol{o}, s)$ and the rewards $r(s, \boldsymbol{o})$ over any concurrent option $\boldsymbol{o}$ defines a semi-Markov decision process [28]. The significance of this result is that SMDP Q-learning methods can be extended to learn to concurrent plans under this model. The extended SMDP Q-learning algorithm for learning to plan with concurrent options updates the multi-option-value function $Q(s, \boldsymbol{o})$ after each decision epoch where the multi-option $\boldsymbol{o}$ is taken in some state s and terminates in s' (under either termination condition):

$$Q(s, \boldsymbol{o}) \leftarrow Q(s, \boldsymbol{o})(1 - \beta) + \beta \left[R + \gamma^k \max_{\boldsymbol{o}' \in O_{s'}} Q(s', \boldsymbol{o}') \right] \quad (2)$$

where k denotes the number of time steps between initiation of the multi-option $\boldsymbol{o}$ in state s and its termination in state s', and R denotes the cumulative discounted reward over this period. The result of using this algorithm on the simple grid world problem in shown in Figure 7. The figure illustrates the difference in performance under different termination conditions (T_{all}, T_{any}, and T_{cont}).

The concurrent option model can be extended to allow cases when options executing in parallel modify the same shared variables at the same time. Using a DBN representation of concurrent actions, the above theorem continues to hold in this case (and the concurrent SMDP Q-learning method works as well).

4.2 Learning Multiagent Task-Level Coordination Strategies

The second case study uses hierarchical abstraction to learn multiagent coordination strategies. Figure 8 illustrates a robot trash collection task, where the two agents $A1$ and $A2$ will maximize their performance at the task if they learn to coordinate with each other. Here, we want to design learning algorithms for *cooperative* multiagent tasks [42], where the agents learn the coordination skills by trial and error. The key idea here is that coordination skills are learned more efficiently if agents learn to synchronize using a hierarchical representation of the task structure [32]. In particular, rather than each robot learning its response to low-level primitive actions of the other robots (for instance, if A1 goes forward, what should A2 do), they learn high-level coordination knowledge (what is the utility of A2 picking up trash from T1 if A1 is also picking up from the same

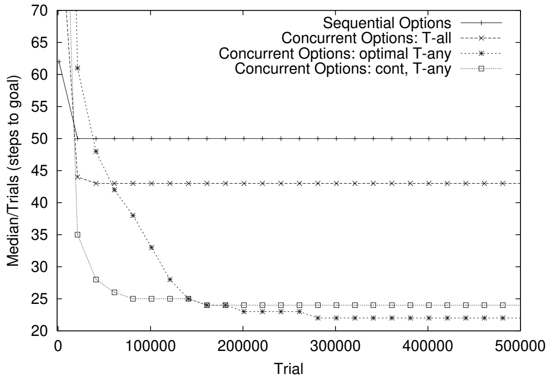

Fig. 7. This graph compares an SMDP technique for learning concurrent plans (under various termination conditions) with a slower "get-to-door-then-pickup-key" sequential plan learner. The concurrent learners outperform the sequential learner, but the choice of termination affects the speed and quality of the final plan.

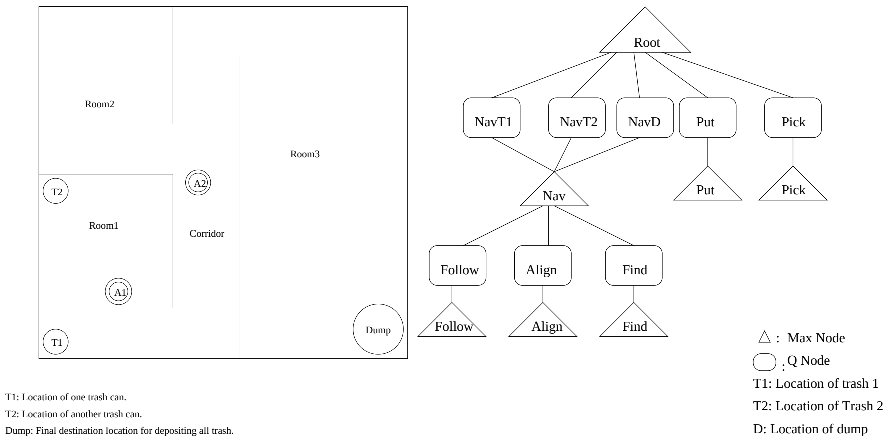

Fig. 8. A two-robot ($A1$ and $A2$) trash collection task. The robots can learn to coordinate much more rapidly using the task structure than if they attempted to coordinate at the level of primitive movements.

bin, and so on). The proposed approach differs significantly from previous work in multiagent reinforcement learning [15, 35] in using hierarchical task structure to accelerate learning, and as well in its use of concurrent temporally extended actions.

One general approach to learning task-level coordination is to extend the above concurrency model to the joint state action space, where base level policies remain fixed. However, an extension of this approach is now presented, where agents learn coordination skills and the base-level policies all at once. However, convergence to optimal (hierarchical) policies is no longer assured since lower level (subroutine) policies are varying at the same time when learning higher level routines. The ideas extend to other formalisms also, but for the sake of clarity, we focus on the MAXQ value function decomposition approach [5]. This decomposition is based on storing the value function in a distributed manner across all nodes in a task graph. The value function is computed on demand by querying lower level (subtask) nodes whenever a high level (task) node needs to be evaluated.

It is necessary to generalize the MAXQ decomposition from its original sequential single-agent setting to the concurrent multiagent coordination problem. Let $\boldsymbol{o} = (\boldsymbol{o_1}, ..., \boldsymbol{o_n})$ denote a concurrent action, where $\boldsymbol{o_i}$ is the set of concurrent subprocesses under the control of agent i. Let $\boldsymbol{s} = (s_1, ..., s_n)$ denote the joint state. The joint action value function $Q(p, \boldsymbol{s}, \boldsymbol{o})$ represents the value of concurrent action $\boldsymbol{o}$ in joint state $\boldsymbol{s}$, in the *context* of doing parent task p.

The MAXQ decomposition of the Q-function relies on a key principle: the reward function for the parent task is essentially the value function of the child subtask. This principle can be extended to joint concurrent action values as shown below. Define the *completion function* for agent j as $C^j(p, s_j, \boldsymbol{o})$ as the expected cumulative discounted (or average-adjusted) reward of agent j completing concurrent subtask $\boldsymbol{o_j}$ in the context of doing parent task p, when the other agents are performing concurrent subtasks $o_k, \forall k \in \{1, ..., n\}, k \neq j$. The joint concurrent action value function $Q(p, \boldsymbol{s}, \boldsymbol{o})$ is now approximated by each agent j (given only its local state s_j) as:

$$Q^j(p, s_j, \boldsymbol{o}) \approx V^j(\boldsymbol{o_j}, s_j) + C^j(p, s_j, \boldsymbol{o}) \tag{3}$$

$$V^j(p, s_j) = \begin{cases} \max_{\boldsymbol{o_k}} Q^j(p, s_j, \boldsymbol{o_k}) & \text{if parent task p is composite} \\ \sum_{s'_j} P(s'_j \mid s_j, p) R(s'_j \mid s_j, p) & \text{if p is a primitive action} \end{cases}$$

The first term in the $Q(p, s_j, \boldsymbol{o})$ expansion above refers to the discounted sum of rewards received by agent j for doing concurrent action $\boldsymbol{o_j}$ in state s_j. The second term "completes" the sum by accounting for rewards earned for completing the parent task p after finishing $\boldsymbol{o_j}$. The completion function is updated from sample values using an SMDP learning rule. Note that the correct action value is approximated by only considering local state s_j and also by ignoring the effect of concurrent actions $\boldsymbol{o_k}, k \neq j$ by other agents when agent j is performing $\boldsymbol{o_j}$. In practice, a human designer can configure the task graph to store joint concurrent action values at the highest (or lower than the highest as needed) level(s) of the hierarchy as needed.

To illustrate the use of this decomposition in learning multiagent coordination, for the two-robot trash collection task, if the joint action-values are restricted to only the highest level of the task graph under the root, we get the following value function decomposition for agent $A1$:

$$Q^1(Root, s_1, (NavT1, NavT2)) \approx V_t^1((NavT1), s_1) + C_t^1(Root, s_1, (NavT1, NavT2))$$

which represents the value of agent $A1$ doing task $NavT1$ in the context of the overall *Root* task, when agent $A2$ is doing task $NavT2$. Note that this value is decomposed into the value of agent $A1$ doing $NavT1$ subtask itself and the completion sum of the remainder of the overall task done by both agents. In this example, the multiagent MAXQ decomposition embodies the constraint that the value of $A1$ navigating to trash bin $T1$ is independent of whatever $A2$ is doing.

4.3 Hierarchical Memory

When agents learn to act concurrently in real-world environments, the true state of the environment is usually hidden. To address this issue, we need to combine the above methods for learning concurrency and coordination with methods for estimating (joint) hidden state and actions. We have explored two multiscale memory models [9, 39]. *Hierarchical Suffix Memory* (HSM) [9] generalizes the suffix tree model [18] to SMDP-based temporally extended actions. Suffix memory constructs state estimators from finite chains of observation-action-reward triples. In addition to extending suffix models to SMDP actions, HSM also uses multiple layers of temporal abstraction to form longer-term memories at more abstract levels. Figure 9 illustrates this idea for robot navigation for the simpler case of a linear chain, although the tree-based model has also been investigated. An important side-effect is that the agent can look back many steps back in time while ignoring the exact sequence of low-level observations and actions that transpired. Tests in a robot navigation domain showed that HSM outperformed "flat" suffix tree methods, as well as hierarchical methods that used no memory [9]. POMDPs are theoretically more powerful than finite memory models,

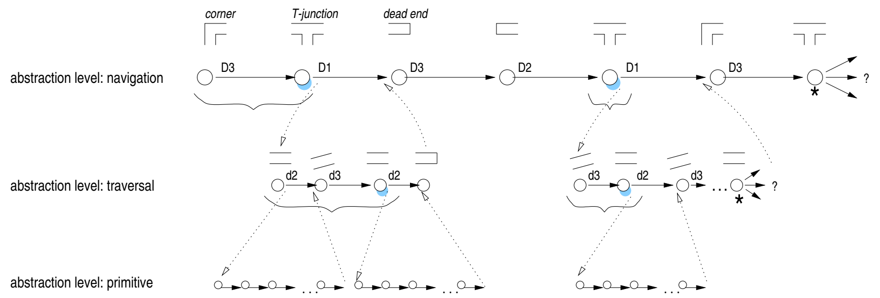

Fig. 9. A hierarchical suffix memory state estimator for a robot navigation task. At the abstract (navigation) level, observations and decisions occur at intersections. At the lower (corridor-traversal) level, observations and decisions occur within the corridor. At each level, each agent constructs states out of its past experience with similar history (shown with shadows).

but past work on POMDPs has mostly studied "flat" models for which learning and planning algorithms scale poorly with model size. We have developed a new *hierarchical POMDP* framework termed H-POMDPs (see Figure 10) [39],

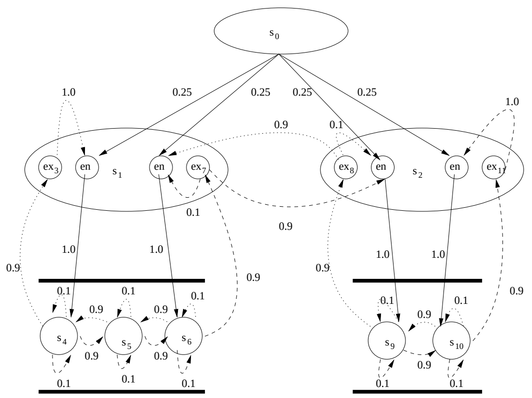

Fig. 10. An example hierarchical POMDP model representing two adjacent corridors in a robot navigation task. The model has two primitive actions, "go-left" indicated with the dotted arrows and "go-right" indicated with the dashed arrows. This HPOMDP has two (unobservable) abstract states s_1 and s_2 and each abstract state has two entry and two exit states. The (hidden) product state s_4, s_5, s_6, s_9, and s_{10} have associated observation models.

by extending the hierarchical hidden Markov model (HHMM) [6] to include rewards and (temporally extended) actions. We have developed a hierarchical EM algorithm for learning the parameters of an H-POMDP model from sequences of observations and actions. Extensive tests on a robot navigation domain show learning and planning performance is much improved over flat POMDP models [39, 38]. The hierarchical EM-based parameter estimation algorithm scales more gracefully to large models because previously learned sub-models can be reused when learning higher levels. Also, the effect of temporally extended actions in H-POMDPs (e.g. exit the corridor) can be modeled at abstract and product level states, which supports planning at multiple levels of abstraction. H-POMDPs have an inherent advantage in allowing belief states to be computed at different levels of the tree. In addition, there is often less uncertainty at higher levels (e.g., a robot is more sure of which corridor it is in, rather than exactly which low level state). A number of heuristics for mapping belief states to action provide good performance in robot navigation (e.g, the most-likely-state (MLS) heuristic assumes the agent is in the state corresponding to the "peak" of the belief state distribution) [13, 31, 22]. Such heuristics work much better in H-POMDPs because they can be applied at multiple levels, and belief states over abstract states usually have lower entropy (see Figure 11). For a detailed study of the H-POMDP model, as well as its application to robot navigation, see [37].

5 Conclusions

In this paper, I described some general approaches to solving large sequential process models through spatiotemporal abstraction. In particular, I presented a

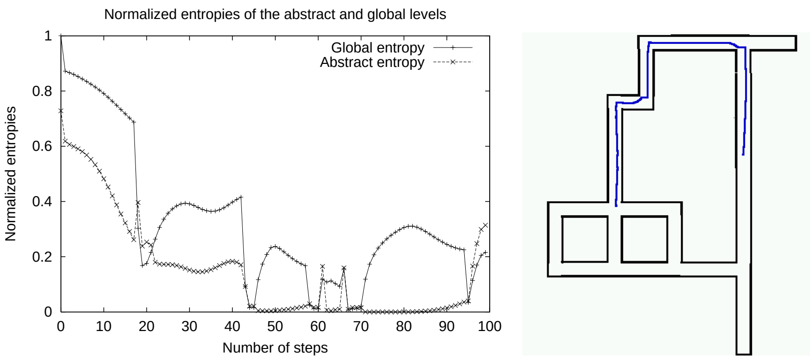

Fig. 11. This plot shows a sample robot navigation run whose trace is on the right, where positional uncertainty (measured by belief state entropy) at the abstract (corridor) level is less than at the product state level. Spatiotemporal abstraction reduces the uncertainty and requires less frequent decision-making, allowing the robot to get to goals without inital positional information.

framework for learning parallel plans that combined factored state representations with temporally extended actions. In addition, I described how this concurrency framework could be extended to learn multiagent coordination strategies by using the overall task hierarchy. Finally, I showed how abstraction can help alleviate the problem of hidden state by building multiresolution state estimators. These case studies are some inital steps towards a more unified and rigorous approach to abstraction of sequential processes.

6 Acknowledgements

I am indebted to my current and former graduate students whose work is summarized in this paper, including Natalia Hernandez-Gardiol, Mohammad Ghavamzadeh, Rajbala Makar, Silviu Minut, Khashayar Rohanimanesh, and Georgios Theocharous. This research was supported in part by grants from the National Science Foundation (KDI), the Defense Advanced Research Projects Agency (MARS and DR programs), Michigan State University, and the University of Massachusetts, Amherst.

References

1. C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000.
2. R.H. Crites and A.G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235–262, 1998.
3. T. Dean and R. Givan. Model minimization in markov decision processes. *Proceedings of AAAI*, 1997.

4. T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
5. T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *International Journal of Artificial Intelligence Research*, 13:227–303, 2000.
6. S. Fine, Y. Singer, and N. Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1), July 1998.
7. Dayne Freitag and Andrew Kachites McCallum. Information extraction with hmms and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Informatino Extraction*, 1999.
8. M. Ghavamzadeh and S. Mahadevan. Continuous-time hierarchical reinforcement learning. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.
9. N. Hernandez and S. Mahadevan. Hierarchical memory-based reinforcement learning. *Proceedings of Neural Information Processing Systems*, 2001.
10. F. Jellinek. *Statistical Methods in Speech Recognition*. MIT Press, 2000.
11. K. Karplus, C. Barrett, and R. Hughey. Hidden markov models for detecting remote protein homologies, 1998.
12. Craig A. Knoblock. An analysis of ABSTRIPS. In James Hendler, editor, *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS 92)*, pages 126–135, College Park, Maryland, USA, 1992. Morgan Kaufmann.
13. S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In D. Kortenkamp, P. Bonasso, and Murphy. R., editors, *AI-based Mobile Robots: Case-studies of Successful Robot Systems*. MIT Press, 1997.
14. Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured mdps. *16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1332–1339, 1999.
15. M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.
16. S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992. Appeared originally as IBM TR RC16359, Dec 1990.
17. S. Mahadevan, N. Marchalleck, T. Das, and A. Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proc. 14th International Conference on Machine Learning*, pages 202–210. Morgan Kaufmann, 1997.
18. Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.
19. N. Meuleau, M. Hauskrecht, K. Kim, L. Peshkin, L. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled markov decision processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1998.
20. S. Minut and S. Mahadevan. A reinforcement learning model of selective visual attention. In *Fifth International Conference on Autonomous Agents*, 2001.
21. K. Murphy and M. Paskin. Linear time inference in hierarchical hmms. *Proceedings of Neural Information Processing Systems*, 2001.
22. I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish: An office-navigation robot. *AI Magazine*, 16(2):53–60, 1995.

23. R.E. Parr. *Hierarchical Control and Learning for Markov Decision Processes.* PhD Thesis, University of California, Berkeley, 1998.
24. J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to POMDP planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML 2001)*, Williams College, MA, June 2001.
25. A. Prieditis. Machine discovery of admissible heuristics, 1993.
26. M. L. Puterman. *Markov Decision Processes.* Wiley Interscience, New York, USA, 1994.
27. B. Ravi and A. Barto. Model minimization in hierarchical reinforcement learning. In *Symposium on Abstraction and Reformulation (SARA 2002).* Springer Verlag, 2002.
28. K. Rohanimanesh and S. Mahadevan. Decision-theoretic planning with concurrent temporally extended actions. In *17th Conference on Uncertainty in Artificial Intelligence*, 2001.
29. K. Rohanimanesh and S. Mahadevan. Incremental learning of factorial markov decision processes. under preparation, 2002.
30. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1994.
31. Hagit Shatkay and Leslie Pack Kaelbling. Learning topological maps with weak local odometric information. In *IJCAI (2)*, pages 920–929, 1997.
32. T. Sugawara and V. Lesser. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, 33:129–154, 1998.
33. R. Sutton and A. Barto. *An introduction to reinforcement learning.* MIT Press, Cambridge, MA., 1998.
34. R. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.
35. M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.
36. Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–278, 1992.
37. G. Theocharous. *Hierarchical Learning and Planning in Partially Observable Markov Decision Processes.* PhD Thesis, Michigan State University, 2002.
38. G. Theocharous and S. Mahadevan. Approximate planning with hierarchical partially observable markov decision processs for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.
39. G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning hierarchical partially observable markov decision processs for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001.
40. G. Wang and S. Mahadevan. Hierarchical optimization of policy-coupled semi-Markov decision processes. In *Proc. 16th International Conf. on Machine Learning*, pages 464–473. Morgan Kaufmann, San Francisco, CA, 1999.
41. C. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, England, 1989.
42. G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press, Cambridge, MA., 1999.