LEARNING AND APPROXIMATE DYNAMIC PROGRAMMING

Scaling Up to the Real World

LEARNING AND APPROXIMATE DYNAMIC PROGRAMMING

Scaling Up to the Real World

Edited by

Jennie Si, Andy Barto, Warren Powell, and Donald Wunsch

A Wiley-Interscience Publication

JOHN WILEY & SONS

New York • Chichester • Weinheim • Brisbane • Singapore • Toronto

Contents

1 Hierarchical Approaches to Concurrency, Multiagency, and Partial Observability

SRIDHAR MAHADEVAN,
MOHAMMAD GHAVAMZADEH,
KHASHAYAR ROHANIMANESH
Computer Science Department
(mahadeva,mgh,khash)@cs.umass.edu
University of Massachusetts, Amherst

GEORGIOS THEOCHAROUS
MIT A. I. Laboratory
theochar@ai.mit.edu
Cambridge, MA

Editor's Summary:
In this chapter the authors summarize their research in hierarchical probabilistic models for decision making involving concurrent action, multiagent coordination, and hidden state estimation in stochastic environments. A hierarchical model for learning concurrent plans is first described for observable single agent domains, which combines compact state representations with temporal process abstractions to determine how to parallelize multiple threads of activity. A hierarchical model for multiagent coordination is then presented, where primitive joint actions and joint states are hidden. Here, high level coordination is learned by exploiting overall task structure, which greatly speeds up convergence by abstracting from low level steps that do not need to be synchronized. Finally, a hierarchical framework for hidden state estimation and action is presented, based on multi-resolution statistical modeling of the past history of observations and actions.

1.1 INTRODUCTION

Despite five decades of research on models of decision-making, artificial systems remain significantly below human level performance in tasks involving the planning and execution of concurrent actions, tasks where perceptual limitations require remembering past observations, and finally problems where the behavior of other agents needs to be taken into account. Driving (see Figure 1.1) is one of many human

activities that involve simultaneously grappling with all these challenges. To date, a general framework that jointly addresses concurrency, multiagent coordination, and hidden state estimation has yet to be developed, but some of the essential components of such a framework are beginning to be understood. In this chapter, we provide a broad overview of our previous research on hierarchical models of concurrency, multiagency, and partial observability.

Humans learn to carry out multiple concurrent activities at many abstraction levels, when acting alone or in concert with other humans. Figure 1.1 illustrates a familiar everyday example, where drivers learn to observe road signs and control steering, but also manage to engage in other activities such as operating a radio, or carrying on a cellphone conversation. Concurrent planning and coordination is also essential to many important engineering problems, such as flexible manufacturing with a team of machines to scheduling robots to transport parts around factories. All these tasks involve a hard computational problem: how to sequence multiple overlapping and interacting parallel activities to accomplish long-term goals. The problem is difficult to solve in general since it requires learning a mapping from noisy incomplete perceptions to multiple temporally extended decisions with uncertain outcomes. It is an impressive feat that humans are able to reliably solve problems such as driving with relatively little effort, and with modest amounts of training.

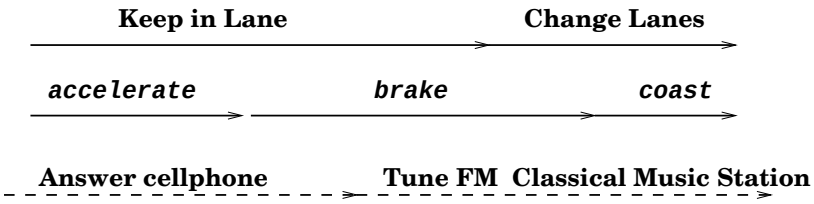

Fig. 1.1 Driving is one of many human activities illustrating the three principal challenges addressed in this chapter: concurrency, multiagency, and partial observability. To drive successfully, humans execute multiple parallel activities,while coordinating with actions taken by other drivers on the road, and use memory to deal with their limited perceptual abilities.

In this chapter, we summarize our past research on a hierarchical approach to concurrent planning and coordination in stochastic single agent and multiagent environments. The overarching theme is that efficient solutions to these challenges can be developed by exploiting multi-level temporal and spatial abstraction of actions and states. The framework will be elaborated in three parts. First, a hierarchical model for learning concurrent plans is presented, where for simplicity, it is assumed that agents act alone, and can fully observe the state of the underlying process. The key idea here is that by combining compact state representations with temporal process abstractions, agents can learn to parallelize multiple threads of activity. Next, a hierarchical model for multiagent coordination is described, where primitive joint actions and joint states may be hidden. This partial observability of lower level actions is indeed a blessing, since it allows agents to speedup convergence by abstracting from low-level steps that do not need to be synchronized. Finally, we present a hierarchi-

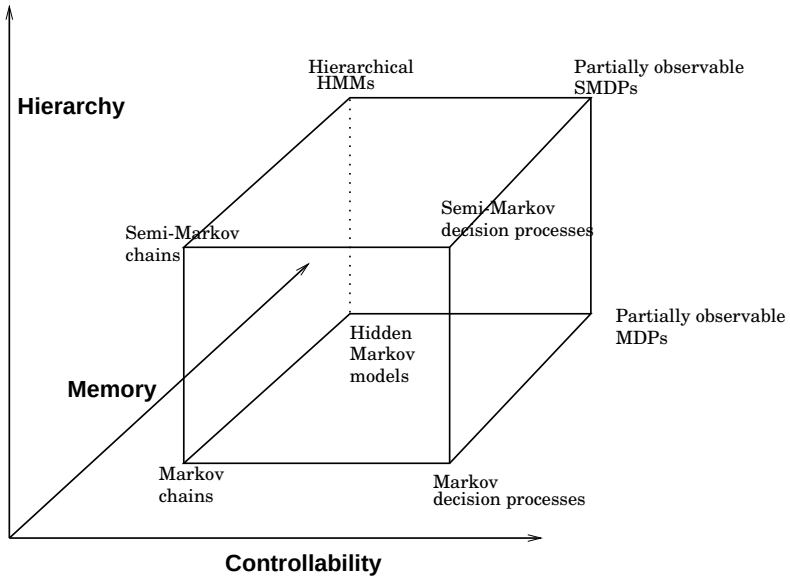

Fig. 1.2 A spectrum of Markov process models along several dimensions: whether agents have a choice of action, whether states are observable or hidden, and whether actions are unit-time (single-step) or time-varying (multi-step).

cal approach to state estimation, based on multi-resolution statistical modeling of the past history of observations and actions.

The proposed approaches all build on a common *Markov decision process* (MDP) modeling paradigm, which is summarized in the next section. Previous MDP-based algorithms have largely focused on sequential compositions of closed-loop programs. Also, earlier MDP-based approaches to learning multiagent coordination ignored hierarchical task structure, resulting in slow convergence. Previous finite memory and partially observable MDP-based methods for state estimation used flat representations, which scale poorly to long experience chains and large state spaces. The algorithms summarized in this chapter address these limitations in previous work, by using new spatiotemporal abstraction based approaches for learning concurrent closed-loop programs and abstract task-level coordination, in the presence of significant perceptual limitations.

1.2 BACKGROUND

Probabilistic finite state machines have become a popular paradigm for modeling sequential processes. In this representation, the interaction between an 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, a state is 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 [36], decision-theoretic planning [2], information retrieval [8], speech recognition [11], active vision [22], and robot navigation [14]. In this chapter, we are interested in abstracting sequential Markov processes using two strategies: state aggregation/decomposition and temporal abstraction. State decomposition methods typically represent states as collections of *factored* variables [2], or simplify the automaton by eliminating "useless" states [4]. Temporal abstraction mechanisms, for example in hierarchical reinforcement learning [37, 6, 25], 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 [29].

Figure 1.2 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 chapter, we will provide brief descriptions of many of these models below, beginning in this section with the basic MDP model.

A *Markov decision process* (MDP) [28] 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*):

$$V^*(s) = \max_{a \in A(s)} \left(r(s,a) + \gamma \sum_{s'} P^a_{ss'} V^*(s') \right) \tag{1.1}$$

MDPs have been applied to many real-world domains, ranging from robotics [14, 17] to engineering optimization [3, 18], and game playing [39]. 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 [45]. 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 may also be key to scaling to large problems [15].

1.3 SPATIOTEMPORAL ABSTRACTION OF MARKOV PROCESSES

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

1.3.1 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 a robot navigation task, a "go forward" action might itself be comprised of a lower-level actions for moving through a 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 (for an extended review of SMDPs and hierarchical action models, see [1]). 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 a state transition matrix defining the single-step transition probability of the effect of each action, and R is the reward function. For continuous-time SMDPs, F is a function giving probability of transition times for each state-action pair until *natural termination*. The transitions are at decision epochs only. The SMDP represents snapshots of the system at decision points, whereas the so-called *natural process* [28] describes the evolution of the system over all times. For discrete-time SMDPs, the transition distribution is written as $F(s', N \mid s, a)$, which specifies the expected number of steps N that action a will take before terminating (naturally) in state s' starting in state s. For continuous-time SMDPs, $F(t \mid s, a)$ 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.

Several frameworks for hierarchical reinforcement learning have been proposed, all of which are variants of SMDPs, including options [37], MAXQ [6], and HAMs [25]. We discuss some of these in more detail in the next section.

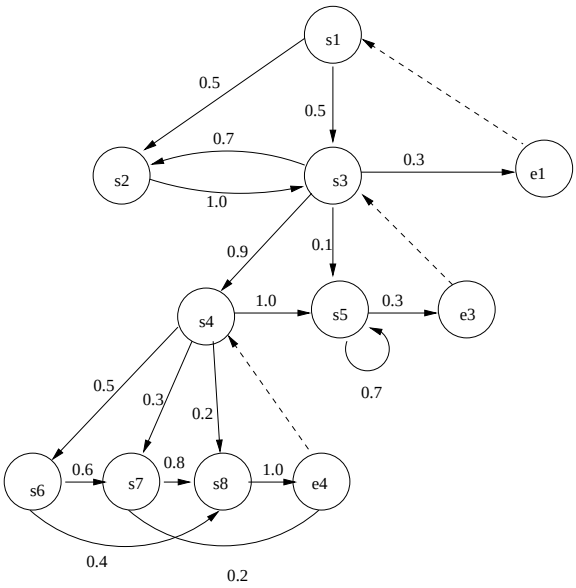

Fig. 1.3 An example hierarchical hidden Markov model. Only leaf nodes produce observations. Internal nodes can be viewed as generating sequences of observations.

1.3.2 Hierarchical Hidden Markov Models

Hidden Markov models (HMMs) are a widely-used probabilistic model for representing time-series data, such as speech [11]. 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 $P(o \mid s)$ as a function of the underlying state s. HMMs have been widely applied to many time-series problems, ranging from speech recognition [11], information extraction [8], and bioinformatics [12]. 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 environments [34], 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 [7]. 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 1.3) 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 1.3) do not produce observations and cannot be activated through a vertical transition from their parent.

Figure 1.3 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 faster inference algorithm for hierarchical HMMs by converting an HHMM into a dynamic Bayes network [23].

1.3.3 Factored 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) [5] 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 action $a \in A$, 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)* $P(X_i^{t+1} \mid \phi(X_i^{t+1}), a)$ in which $\phi(X_i^{t+1})$ denotes the parents of X_i^{t+1} in the graph. The transition probability $P(X^{t+1} \mid X^t, a)$ is then defined by: $P(X^{t+1} \mid X^t, a) = \prod_i^n P(X_i^{t+1} \mid w_i, a)$ where w_i is a vector whose elements are the values of the $X_j^t \in \phi(X_i^{t+1})$.

Figure 1.4 shows a popular toy problem called the Taxi Problem [6] 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 factored MDP with four state variables each taking on values as explained above. Figure 1.4 shows a factorial representation of taxi domain for *Pickup* and *Fillup* actions.

While it is relatively straightforward to represent factored MDPs, it is not easy to solve them because in general the solution (i.e., the optimal value function) is not factored. While a detailed discussion of this issue is beyond the scope of this article, a popular strategy is to construct an approximate factored value function as a linear summation of basis functions (see [15]). The use of factored representations is useful not only in finding (approximate) solutions more quickly, but also in learning a factored transition model in less time. For the taxi task illustrated in Figure 1.4, one idea that we have investigated is to express the factored transition probabilities as a mixed memory factorial Markov model [33]. Here, each transition probability (edge in the graph) is represented a weighted mixture of distributions, where the weights can be learned by an expectation maximization algorithm.

More precisely, the action model is represented as a weighted sum of *cross-transition* matrices:

$$P(x_{t+1}^i \mid X_t, a) = \sum_{j=1}^{n} \psi_a^i(j) \tau_a^{ij}(x_{t+1}^i \mid x_t^j) \tag{1.2}$$

where the parameters $\tau_a^{ij}(x' \mid x)$ are n^2 elementary $k \times k$ transition matrices and parameters $\psi_a^i(j)$ are positive numbers that satisfy $\sum_{j=1}^n \psi_a^i(j) = 1$ for every action $a \in A$ (here, $0 \leq i, j \leq n$, where n is the number of state variables). The number of free parameters in this representation is $O(An^2k^2)$ as opposed to $O(Ak^{2n})$ in the non-compact case. The parameters $\psi_a^i(j)$ measure the contribution of different state variables in the previous time step to each state variable in the current state. If the problem is completely factored, then $\psi^i(j)$ is the identity matrix whose i^{th} component is independent of the rest. Based on the amount of factorization that exists in an environment, different components of $\psi_a^i(j)$ at one time step will influence the

i^{th} component at the next. The cross-transition matrices $\tau_a^{ij}(x' \mid x)$ provide a compact way to parameterize these influences.

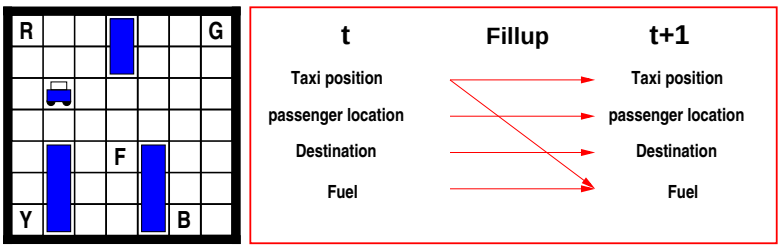

Fig. 1.4 The taxi domain is an instance of a factored Markov process, where actions such as fillup can be represented compactly using dynamic Bayes networks.

Figure 1.5 shows the learning of a factored MDP compared to a table-based MDP, averaged over 10 episodes of 50000 steps. Each point on the graph represents the RMS error between the learned model and the ground truth, averaged over all states and actions. The FMDP model error drops quickly in the early stages of learning. Theoretically, the tabular maximum likelihood approach (which estimates each transition probability as the ratio of transitions between two states versus the number of transitions out of a state) will eventually learn the the exact model if every pair of states and action are executed infinitely often. However, the factored approach, which uses a mixture weighted representation, is able to generalize much more quickly to novel states.

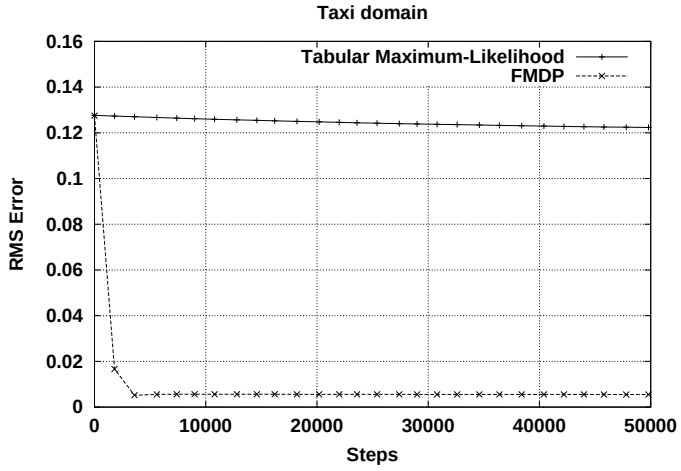

Fig. 1.5 Comparing factored versus tabular model learning performance in the taxi domain.

1.3.4 Structural Decomposition of Markov Processes

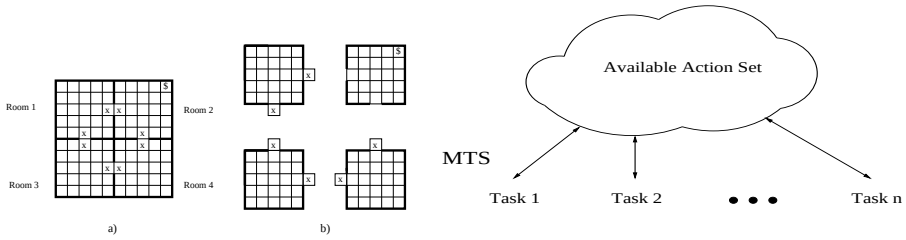

Fig. 1.6 State and action-based decomposition of Markov processes.

Other related techniques for decomposition of large MDPs have been explored, and some of these are illustrated in Figure 1.6. A simple decomposition strategy is to split a large MDP into sub-MDPs, which interact "weakly" [4, 25, 37]. 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 [21], or in conversational robotics [26]. 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 [44].

1.4 CONCURRENCY, MULTIAGENCY, AND PARTIAL OBSERVABILITY

This section summarizes our recent research on exploiting spatiotemporal abstraction to produce improved solutions to three difficult problems in sequential decision-making: learning plans involving concurrent action, multiagent coordination, and using memory to estimate hidden state.

1.4.1 A Hierarchical Framework for Concurrent Action

We now describe a probabilistic model for learning concurrent plans over temporally extended actions [30, 31]. The notion of concurrent action is formalized in a general way, to capture both situations where a single agent can execute multiple parallel processes, as well as the multi-agent case where many agents act in parallel.

The *Concurrent Action Model (CAM)* is defined as $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of *primary* actions, $\mathcal{T}$ is a transition probability distribution $\mathcal{S} \times \mathcal{P}(\mathcal{A}) \times \mathcal{S} \times \mathbf{N} \rightarrow [0, 1]$, where $\mathcal{P}(\mathcal{A})$ is the power-set of the primary actions and $\mathbf{N}$ is the set of natural numbers, and $\mathcal{R}$ is the reward function mapping $\mathcal{S} \rightarrow \Re$.

Here, a concurrent action is viewed as a set of primary actions (hereafter called a *multi-action*), where each primary action is either a single step action, or a *temporally extended action* (e.g., modeled as a closed loop policy over single step actions [37]). Figure 1.7 illustrates a toy example of concurrent planning. The general problem is as follows. The agent is given a set of primary 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 primary 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) 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) [5].

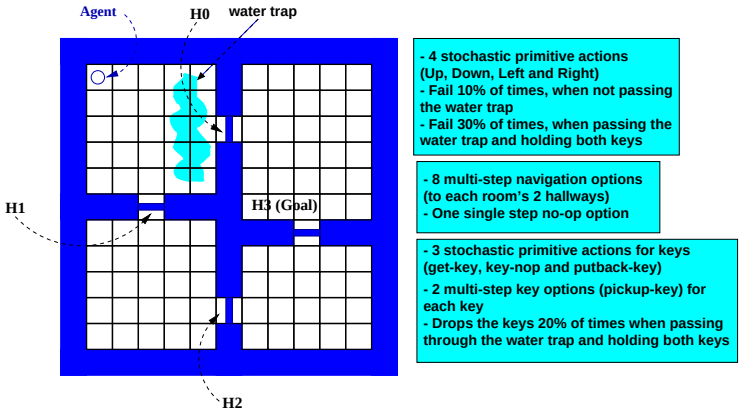

Fig. 1.7 A grid world problem to illustrate concurrent planning: the agent is given subroutines for getting to each door from any interior room state, and for opening a locked door. It has to learn the shortest path to the goal by concurrently combining these subroutines. The agent can reach the goal 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.

Since multiple concurrent primary 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 (Figure 1.8, left). Alternatively, a decision epoch can be defined to occur when all actions currently running terminate, which we refer to as the T_{all} condition (Figure 1.8, middle). We can design other termination schemes by combining T_{any} and T_{all} : for example, another termination scheme called $T_{continue}$ is one that always terminates based on the T_{any} termination scheme, but allows those

primary actions that did not terminate naturally to keep executing, while initiating new primary actions if they are going to be useful (Figure 1.8, right).

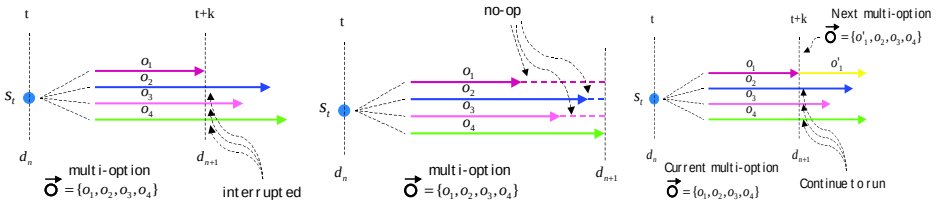

Fig. 1.8 Left: T_{any}termination scheme. Middle: T_{all} termination scheme. Right: $T_{continue}$ termination scheme.

For concreteness, we will describe the concurrent planning framework when the primary actions are represented as *options* [37]. The treatment here is restricted to options over discrete-time SMDPs and deterministic policies, but the main ideas extend readily to other hierarchical formalisms [6, 25] and to continuous-time SMDPs [9, 28]. More formally, an option o consists of three components: a policy $\pi : S \rightarrow A$, a termination condition $\beta : S \rightarrow [0, 1]$, 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 1.7, 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 primary actions 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 a given 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 $p(s' \mid s, o) = \sum_{k=1}^{\infty} p^o(s, s', k)\gamma^k$ (note that when $\gamma < 1$, the transition model is not a stochastic matrix, since the distributions do not sum to 1). 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 1.1. Under either definition of the termination event (i.e.; T_{any} , T_{all} , and $T_{continue}$), 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-actions, and executes each one until its termination according to a given termination condition forms a semi-Markov decision process.

The proof requires showing that the state transition dynamics $p(s', N \mid \vec{a}, s)$ and the rewards $r(s, \vec{a})$ over any concurrent action $\vec{a}$ defines a semi-Markov decision process [30]. The significance of this result is that SMDP Q-learning methods can be extended to learn concurrent plans under this model. The extended SMDP Q-learning algorithm for learning to plan with concurrent actions updates the multi-action-value function $Q(s, \vec{a})$ after each decision epoch where the multi-action $\vec{a}$ is taken in some state s and terminates in s' (under a specific termination condition):

$$Q(s, \vec{a}) \leftarrow Q(s, \vec{a})(1 - \beta) + \beta \left[R + \gamma^k \max_{\vec{a'} \in O_{s'}} Q(s', \vec{a'}) \right] \tag{1.3}$$

where k denotes the number of time steps between initiation of the multi-action $\vec{o}$ 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 1.9. The figure illustrates the difference in performance under different termination conditions (T_{all}, T_{any}, and T_{cont}).

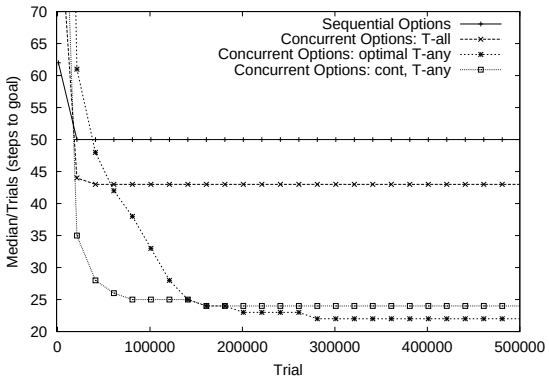

Fig. 1.9 This graph compares an SMDP technique for learning concurrent plans (under various termination conditions) with a slower "get-to-door-then-pickup-key" sequential plan learner. The concurrent learners outperform the sequential learner, but the choice of termination affects the speed and quality of the final plan.

The performance of the concurrent action model also depends on the termination event defined for that model. Each termination event trades-off between the optimality of the learned plan and how fast it converges to its optimal policy. Let π^{*seq}, π^{*all} and π^{*any} denote the optimal policy when the primary actions are executed sequentially;

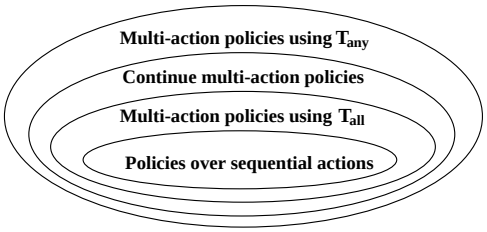

Fig. 1.10 Comparison of policies over multi-actions and sequential primary actions using different termination schemes.

when termination construct T_{all} is used; and when termination construct T_{any} is used, respectively. Also let $\pi_{continue}$ represent the policy learned based on the $T_{continue}$ termination construct. Intuitively, the models with a termination construct that imposes more frequent multi-action termination (such as T_{any} and $T_{continue}$), tend to *articulate* more frequently and should perform more optimally. However due to more interruption, they may converge more slowly to their optimal behavior. Based on the definition of each termination construct we can prove the following theorem:

Theorem 2: In a concurrent action model and a set of termination schemes $\{T_{any}, T_{all}, T_{continue}\}$, the following partial ordering holds among the optimal policy based on T_{any} , the optimal policy based on T_{all} , the $T_{continue}$ policy and the optimal sequential policy: $\pi^{*seq} \leq \pi^{*all} \leq \pi_{continue} \leq \pi^{*any}$.

where $\leq$ denotes the partial ordering relation over policies. Figure 1.10 illustrates the results defined by Theorem 2. According to this figure, the optimal multi-action policies based on T_{any} and T_{all} , and also $T_{continue}$ multi-action policies dominate (with respect to the partial ordering relation defined over policies) the optimal policies over the sequential case. Furthermore, policies based on $T_{continue}$ multi-actions dominate the optimal multi-action policies based on T_{all} termination scheme, while themselves being dominated by the optimal multi-action policies based on T_{any} termination scheme.

1.4.2 Learning Multiagent Task-Level Coordination Strategies

The second case study uses hierarchical abstraction to design efficient learning algorithms for *cooperative* multiagent systems [46]. Figure 1.11 illustrates a multiagent automated guided vehicle (AGV) scheduling task, where four AGV agents will maximize their performance at the task if they learn to coordinate with each other. 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 [35]. In particular, rather than each AGV learning its response to low-level primitive actions of the other AGV agents (for instance, if AGV1 goes forward, what should AGV2 do), they learn high-level coordination knowledge (what is the utility of AGV1 delivering

material to machine M3 if AGV2 is delivering assembly from machine M2, and so on). The proposed approach differs significantly from previous work in cooperative multiagent reinforcement learning [3, 38] in using hierarchical task structure to accelerate learning, and as well in its use of concurrent temporally extended actions.

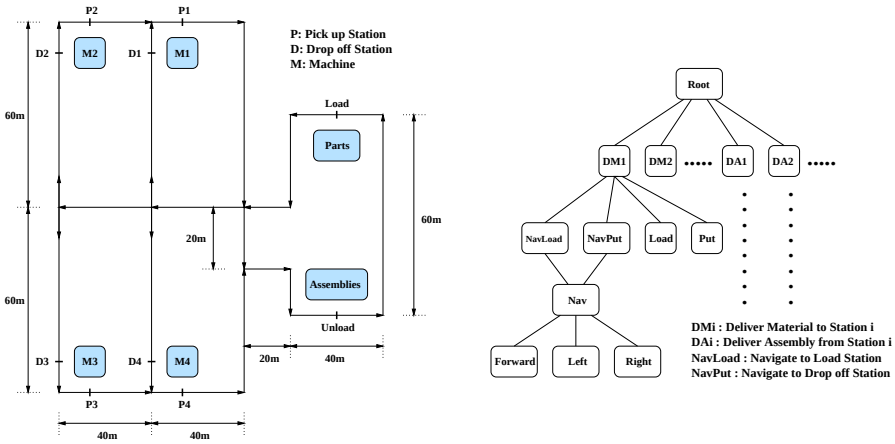

Fig. 1.11 A multiple automated guided vehicle (AGV) optimization task. There are four AGV agents (not shown) which carry raw materials and finished parts between the machines and the warehouse. The task graph of this problem is shown on the right hand side of this figure.

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. An extension of this approach is now presented, where agents learn coordination skills and the base-level policies simultaneously.

The hierarchical multiagent reinforcement learning algorithm described here can be implemented using other hierarchical reinforcement learning formalisms also, but for the sake of clarity, we use the MAXQ value function decomposition approach [6]. 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. The overall task is first decomposed into subtasks up to the desired level of details, and the task graph is constructed. We illustrate the idea using the above multiagent AGV scheduling problem. This task can be decomposed into subtasks and the resulting task graph is shown in figure 1.11. All AGV agents are given the same task graph (homogeneous agents) and need to learn three skills. First, how to do each subtask, such as deliver parts to machine M1 or navigation to drop off station D3, and when to perform load or put action. Second, the agents also need to learn the order to do subtasks (for instance go to pick up station of a machine and pick up an assembly, before heading to the unload station). Finally, the agents also need to learn how to coordinate with other agents (i.e. AGV1 can deliver parts

to machine M4 whereas AGV3 can deliver assemblies from machine M2). We can distinguish between two learning approaches. In the *selfish* case, the agents learn with the given task graph, but make no attempt to coordinate with each other. In the *cooperative* case, coordination skills among agents are learned by using joint actions at the level(s) immediately under the root task. Therefore, it is necessary to generalize the MAXQ decomposition from its original sequential single-agent setting to the concurrent multiagent coordination problem. We call this extension of MAXQ, *cooperative* MAXQ [19]. In this algorithm, each agent learns joint abstract action values by communicating with other agents only the high-level subtasks that they are doing. Since high-level tasks can take a long time to complete, communication is needed only fairly infrequently, which is a significant advantage over flat methods. A further advantage is that agents learn coordination skills at the level of abstract actions and it allows for increased cooperation skills as agents do not get confused by low level details. In addition, each agent has only local state information and is ignorant about the other agent's state. Keeping track of just local information greatly simplifies the underlying reinforcement learning problem. This is based on the idea that in many cases, the state of the other agent might be roughly estimated just by knowing about the high-level action being performed by the other agent.

Let $\vec{s} = (s_1, \ldots, s_n)$ and $\vec{a} = (a_1, \ldots, a_n)$ denote a joint state and a concurrent action, where s_i is the local state and a_i is the action being performed by agent i. Let the joint action value function $Q(p, \vec{s}, \vec{a})$ represents the value of concurrent action $\vec{a}$ in joint state $\vec{s}$, in the context of executing 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. The most salient feature of the *cooperative* MAXQ algorithm, is that the top level(s) (the level immediately below the root and perhaps lower levels) of the hierarchy is (are) configured to store the *completion function* values for joint abstract actions of all agents. The *completion function* $C(p, \vec{s}, \vec{a})$ is the expected cumulative discounted reward of completing parent task p after finishing concurrent action $\vec{a}$, which was invoked in state $\vec{s}$. The joint concurrent value function $V(p, \vec{s})$ is now approximated by each agent i (given only its local state s_i) as:

$$V^i(p, s_i) = \begin{cases} \max_{a_i} Q^i(p, s_i, \vec{a}) & if \text{ p } is\ a\ composite\ action \\ \sum_{s'_i} P(s'_i \mid s_i, p) R(s'_j \mid s_j, p) & if \text{ p } is\ a\ primitive\ action \end{cases}$$

where the action value function of agent i (given only its local state s_i) is defined as

$$Q^i(p, s_i, \vec{a}) \approx V^i(a_i, s_i) + C^i(p, s_i, \vec{a}) \tag{1.4}$$

The first term in equation 1.4, $V^i(a_i, s_i)$, refers to the discounted sum of rewards received by agent i for performing action a_i in local state s_i. The second term, $C^i(p, s_i, \vec{a})$, completes the sum by accounting for rewards earned for completing the parent task p after finishing subtask a_i. The completion function is updated in this algorithm from sample values using an SMDP learning rule. Note that the correct

action value is approximated by only considering local state s_i and also by ignoring the effect of concurrent actions $a_k, k \neq i$ by other agents when agent i is performing action a_i. 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.

To illustrate the use of this decomposition in learning multiagent coordination for the AGV scheduling 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 AGV1:

$$Q^1(Root, s_1, DM3, DA2, DA4, DM1) \approx V^1(DM3, s_1) + C^1(Root, s_1, DM3, DA2, DA4, DM1)$$

which represents the value of AGV1 performing task DM3 in the context of the overall root task, when AGV2, AGV3 and AGV4 are executing DA2, DA4 and DM1. Note that this value is decomposed into the value of AGV1 performing DM3 subtask itself and the completion sum of the remainder of the overall task done by all four agents.

Figure 1.12 compares the performance and speed of the *cooperative* MAXQ algorithm with other learning algorithms, including single-agent MAXQ and selfish multiagent MAXQ, as well as several well-known AGV scheduling heuristics like "first come first serve", "highest queue first" and "nearest station first".

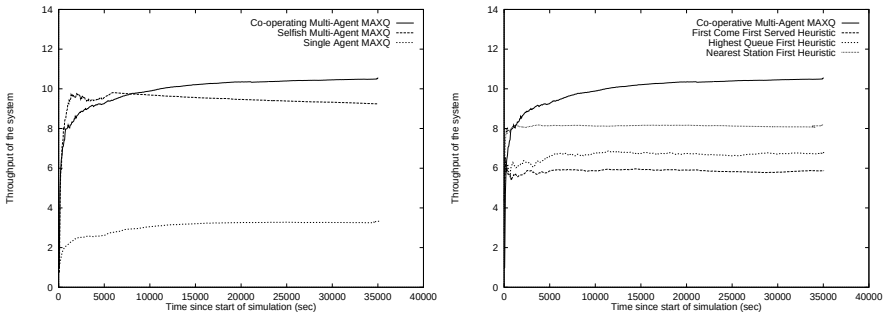

Fig. 1.12 This figure compares the performance of the *cooperative* MAXQ algorithm with other learning methods, including single-agent MAXQ and selfish multiagent MAXQ, as well as several well-known AGV scheduling heuristics. The throughput is measured in terms of the number of finished assemblies deposited at the unload station per unit time.

1.4.3 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 hidden state. We have explored two multiscale memory models [10, 42]. *Hierarchical*

Suffix Memory (HSM) [10] generalizes the suffix tree model [20] 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 1.13 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 [10].

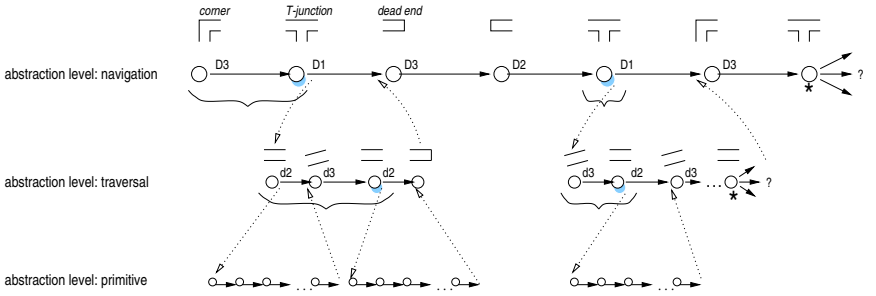

Fig. 1.13 A hierarchical suffix memory state estimator for a robot navigation task. At the abstract (navigation) level, observations and decisions occur at intersections. At the lower (corridor-traversal) level, observations and decisions occur within the corridor. At each level, each agent constructs states out of its past experience with similar history (shown with shadows).

Partially observable MDPs are theoretically more powerful than finite memory models, 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 1.14) [42], by extending the hierarchical hidden Markov model (HHMM) [7] to include rewards, multiple entry/exit points into abstract states and (temporally extended) actions.

H-POMDPs can also be represented as Dynamic Bayesian networks [43], in a similar way that HHMMs can be represented as DBNs [23]. Figure 1.15 shows a Dynamic Bayesian net representation of H-POMDPs. This model differs from the model described in [23] in two basic ways: the presence of action nodes A, and the fact that exit nodes X are no longer binary.

In the particular navigation example shown in Figure 1.14, the exit node X_t can take on five possible values, representing no-exit, north-exit, east-exit, south-exit, and west-exit. If X_t = no-exit, then we make a horizontal transition at the concrete level, but the abstract state is required to remain the same. If $X_t \neq$ no-exit, then we enter a new abstract state; this abstract state then makes a vertical transition into a new

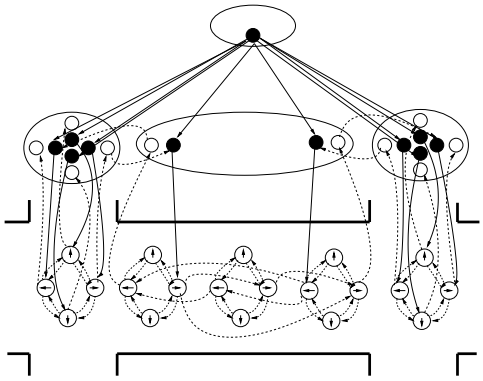

Fig. 1.14 State transition diagram of a hierarchical POMDP used to model corridor environments. Large ovals represent abstract states; the small solid circles within them represent entry states, and the small hollow circles represent exit states. The small circles with arrows represent production states. Arcs represent non-zero transition probabilities as follows: Dotted arrows from concrete states represent concrete horizontal transitions, dashed arrows from exit states represent abstract horizontal transitions, and solid arrows from entry states represent vertical transitions.

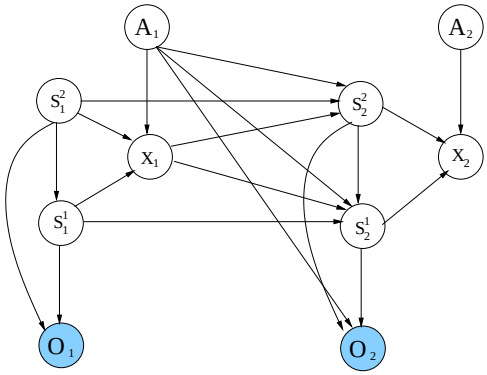

Fig. 1.15 A 2-level HPOMDP represented as a DBN.

concrete state. The new concrete state, S_t^1, depends on the new abstract state, S_t^2, as well as the previous exit state, X_{t-1}. More precisely we can define the conditional probability distributions of each type of node in the DBN as follows: For the abstract nodes,

$$P(S_t^2 = s' \mid S_{t-1}^2 = s, X_{t-1} = x, A_{t-1} = a) = \begin{cases} \delta(s', s) & \text{if } x = \text{no-exit} \\ T^{root}(s'_x \mid s_x, a) & \text{otherwise} \end{cases}$$

where $T^{root}(s'_x \mid s_x, a)$ in the state representation of the HPOMDP model defines the transition probability from abstract state s and exit state x to abstract state s' and entry state x, where x defines the type of entry or exit state (north, east, west, south). S is the parent of s and s' in the state transition model.

For the concrete nodes,

$$P(S_t^1 = s' \mid S_{t-1}^1 = s, S_t^2 = S, X_{t-1} = x, A_{t-1} = a) = \begin{cases} T^S(s' \mid s, a) & \text{if } x = \text{no-exit} \\ V(s' \mid S_x) & \text{otherwise} \end{cases}$$

where $V(s' \mid S_x)$ defines the probability of a vertical transition from abstract state S and entry state of type x to concrete state s'.

For the exit nodes,

$$P(X_t = x \mid S_t^1 = s, S_t^2 = S, A_t = a) = T^S(S_x \mid s, a)$$

where $T^S(S_x \mid s, a)$ is the transition probability from production state s under abstract state S to exit from state S of type x.

For the sensor nodes,

$$P(O_t = z \mid S_t^1 = s, S_t^2 = S, A_{t-1} = a) = O^S(z \mid s, a)$$

where $O^S(z \mid s, a)$ is the probability of perceiving observation z at the sth node under state S after action a.

One of the most important differences of Hierarchical HMMs/POMDPs and flat models are the results of inference. In a hierarchical model a transition to an abstract state at time t is zero, unless the abstract state is able to produce part of the remaining observations and actions in a given sequence. The inference algorithm for the state representation of HHMMs/H-POMDPs in [7], [42] achieves this by doing inference on all possible subsequences of observations under the different abstract states, which leads to $O(K^D T^3)$ time, where K is the number of states at each level of the hierarchy and D is the depth of the hierarchy. In a DBN representation we can achieve the same result as the cubic time algorithms by asserting that the sequence has finished. In our particular implementation we assert that at the last time slice the sequence has finished, and that there is a uniform probability of exit from any of the four orientations. Since we have a DBN representation, we can apply any standard Bayes net inference algorithm, such as junction tree, to perform filtering or smoothing which take in the worse case $O(K^{2D}T)$ time. Empirically it might be less, depending on the size of the cliques being formed, as was shown in [23].

Due to the cubic time complexity of the EM algorithm used in [40] we have developed various approximate training techniques such as “reuse-training”, whereby submodels are trained separately and then combined into an overall hierarchy, and "selective-training" whereby only selected parts of the model are trained for every sequence. Even though these methods require knowledge as to which part of the model the data should be used for, they outperformed the flat EM algorithm in terms of fit to test data, robot localization accuracy, and capability of structure learning at

higher levels of abstraction. However, a DBN-representation allows us to use longer training sequence. In [43] we show how the hierarchical model requires less data for training than the flat model, and also illustrate how combining the hierarchical and factorial representations outperforms both the hierarchical and flat models.

In addition to the advantages over flat methods for model learning, H-POMDPs have an inherent advantage in planning as well. This is because belief states can be computed at different levels of the tree, and 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 temporally extended actions (e.g., move down the corridor) 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) [14, 34, 24]. Such heuristics work much better in H-POMDPs because they can be applied at multiple levels, and probability distributions over abstract states usually have lower entropy (see Figure 1.16). For a detailed study of the H-POMDP model, as well as its application to robot navigation, see [40].

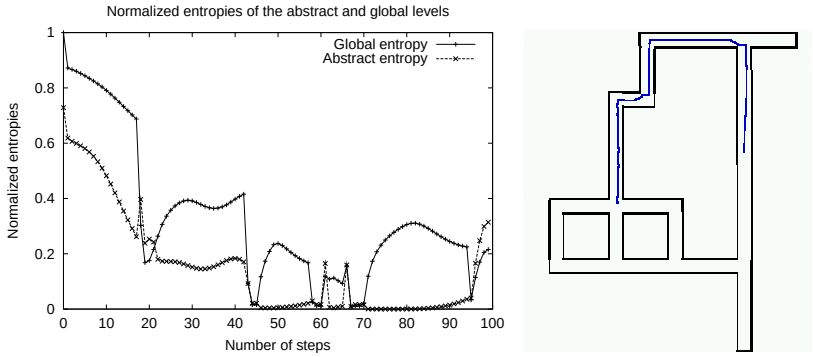

Fig. 1.16 This plot shows a sample robot navigation run whose trace is on the right, where positional uncertainty (measured by belief state entropy) at the abstract (corridor) level is less than at the product state level. Spatiotemporal abstraction reduces the uncertainty and requires less frequent decision-making, allowing the robot to get to goals without initial positional information.

1.5 SUMMARY AND CONCLUSIONS

In this chapter, we presented hierarchical models of decision-making involving concurrent actions, multiagent coordination, and hidden state estimation.The common thread which spanned solutions to these three challenges is that multi-level temporal and spatial abstraction of actions and states can be exploited to achieve effective solutions. The overall approach was presented in three phases, beginning with a hierarchical model for learning concurrent plans for observable single-agent domains. This

concurrency model combined compact state representations with temporal process abstractions to formalize concurrent action. Multiagent coordination was addressed using a hierarchical model where primitive joint actions and joint states are abstracted by exploiting overall task structure, which greatly speeds up convergence since low-level steps are ignored that do not need to be synchronized. Finally, a hierarchical framework for hidden state estimation was presented, which used multi-resolution statistical models of the past history of observations and actions.

Acknowledgments

This research was supported in part by grants from the National Science Foundation (Knowledge and Distributed Intelligence program), the Defense Advanced Research Projects Agency (MARS, Distributed Robotics, and Robot-2020 programs), Michigan State University, and the University of Massachusetts at Amherst.

Bibliography

1. A. Barto, and S. Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete-Event Systems: Theory and Applications*, 13:41–77, 2003.

2. C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000.

3. R.H. Crites and A.G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235–262, 1998.

4. T. Dean and R. Givan. Model minimization in Markov decision processes. *Proceedings of AAAI*, 1997.

5. T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

6. T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *International Journal of Artificial Intelligence Research*, 13:227–303, 2000.

7. S. Fine, Y. Singer, and N. Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1), July 1998.

8. D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Informatino Extraction*, 1999.

9. M. Ghavamzadeh and S. Mahadevan. Continuous-time hierarchical reinforcement learning. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

10. N. Hernandez and S. Mahadevan. Hierarchical memory-based reinforcement learning. *Proceedings of Neural Information Processing Systems*, 2001.

11. F. Jellinek. *Statistical Methods in Speech Recognition*. MIT Press, 2000.

12. K. Karplus, C. Barrett, and R. Hughey. Hidden markov models for detecting remote protein homologies, 1998.

13. C. 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.

14. 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.

15. D. Koller and R. Parr. Computing factored value functions for policies in structured mdps. *16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1332–1339, 1999.

16. 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.

17. 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.

18. 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.

19. R. Makar, M. Ghavamzadeh, and S. Mahadevan. Hierarchical Multiagent Reinforcement Learning. In *Proc. 5th International Conference on Autonomous Agents*, pages 246–253. ACM Press, 2001.

20. A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

21. 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.

22. S. Minut and S. Mahadevan. A reinforcement learning model of selective visual attention. In *Fifth International Conference on Autonomous Agents*, 2001.

23. K. Murphy and M. Paskin. Linear time inference in hierarchical hmms. *Proceedings of Neural Information Processing Systems*, 2001.

24. I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish: An office-navigation robot. *AI Magazine*, 16(2):53–60, 1995.

25. R. Parr. *Hierarchical Control and Learning for Markov Decision Processes*. PhD Thesis, University of California, Berkeley, 1998.

26. 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.

27. A. Prieditis. Machine discovery of admissible heuristics. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 720-725, 1991.

28. M. Puterman. *Markov Decision Processes*. Wiley Interscience, New York, USA, 1994.

29. B. Ravindran and A. Barto. SMDP Homomorphisms: An Algebraic Approach to Abstraction in Semi Markov Decision Processes. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

30. K. Rohanimanesh and S. Mahadevan. Decision-theoretic planning with concurrent temporally extended actions. In *17th Conference on Uncertainty in Artificial Intelligence*, 2001.

31. K. Rohanimanesh and S. Mahadevan. Incremental learning of factorial markov decision processes. under preparation, 2002.

32. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1994.

33. L. Saul and M. Jordan. Mixed Memory Markov Models: Decomposing Complex Stochastic Processes as Mixture of Simpler Ones. *Machine Learning*, *37*, 75–87, 1999.

34. H. Shatkay and L. Kaelbling. Learning topological maps with weak local odometric information. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 920–929, 1997.

35. T. Sugawara and V. Lesser. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, 33:129–154, 1998.

36. R. Sutton and A. Barto. *An introduction to reinforcement learning*. MIT Press, Cambridge, MA., 1998.

37. 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.

38. M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

39. G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–278, 1992.

40. G. Theocharous. *Hierarchical Learning and Planning in Partially Observable Markov Decision Processes*. PhD Thesis, Michigan State University, 2002.

41. 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.

42. 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.

43. G. Theocharous, K. Murphy, and L. Kaelbling. Representing hierarchical POMDPs as DBNs for multi-scale robot localization, In *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, 2003.

44. 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.

45. C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

46. G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA., 1999.