University of Alberta

Library Release Form

Name of Author: Cosmin Păduraru

Title of Thesis: Planning with Approximate and Learned Models of Markov Decision Processes

Degree: Master of Science

Year this Degree Granted: 2007

Cosmin Păduraru
Apt 215, 8515 112 Street
Edmonton, AB
Canada, T6G 1K7

Date: _____

University of Alberta

PLANNING WITH APPROXIMATE AND LEARNED MODELS OF MARKOV DECISION PROCESSES

by

Cosmin Păduraru

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Winter 2007

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Planning with Approximate and Learned Models of Markov Decision Processes** submitted by Cosmin Păduraru in partial fulfillment of the requirements for the degree of **Master of Science**.

_____

Dr. Richard S. Sutton
Co-Supervisor

_____

Dr. Vadim Bulitko
Co-Supervisor

_____

Dr. Michael Bowling

_____

Dr. Bernard Linsky
External Examiner

Date: _____

To my amazingly supportive parents

Abstract

Planning, the process of using a model of the world to compute a policy for selecting actions, is a key component of artificial intelligence. Planning in realistic domains poses many challenges, such as dealing with large problem sizes, non-deterministic effects of actions or *a priori* unknown dynamics. A planning system that addresses these challenges must represent the model compactly, using function approximation, deal with stochastic action effects, and learn the model from experience. Existing methods for planning with approximate and stochastic models, however, make restrictive assumptions about the world's structure. In this thesis, a sampling-based planning method with general function approximation for the stochastic model will be proposed as a less restrictive alternative. Experiments in a continuous, stochastic domain show that the proposed method can be more data-efficient than a model-free alternative. In addition, preliminary theoretical results suggest that, for linear function approximators, an approximate model that only represents expected values may be sufficient for planning. The soundness of planning with approximate models is supported by the general theoretical results in Chapter 3.

Acknowledgements

I am extremely grateful for all the support and encouragement I have received throughout my M.Sc. work at the University of Alberta. The university, and the Department of Computing Science in particular, have created a wonderful research environment. I have learned immensely from my two supervisors, Rich Sutton and Vadim Bulitko, and most other members of the RLAI and IRCL research groups have helped me along the way in one way or another. In particular, Mark Ring, Csaba Szepesvari, Mohammad Ghavamzadeh, Anna Koop, David Silver, Adam White and Doina Precup have dedicated a significant portion of their time to helping me with the research and writing that this thesis is a result of. I would also like to thank Michael Bowling and Bernard Linsky for finding the time to serve as defense committee members in a very busy period of the year. Finally, I have to thank Iulian Radu, from whom I learned the basics of artificial intelligence many years ago.

Table of Contents

Chapter 1

Introduction

Good poker players are able to exploit knowledge of an opponent's strategy; a driver that knows the city well is able to navigate between any two places; engineers can design new, complex structures because they know how basic parts and materials interact. In all of these examples knowledge of the world is used by a decision maker in order to achieve some objective. In artificial intelligence (AI) this process is generally referred to as 'planning', and it has long been a subject of investigation for AI researchers. The aim of this thesis is to make advances in the design and analysis of a particular class of planning methods, more specifically methods that use approximate and learned world models.

Designing efficient planning methods is important because they allow artificial intelligence agents to make use of a model (either provided or learned). Using a model is particularly advantageous for agents that have to perform a variety of tasks in the same setting. This would be the case, for instance, when driving between different locations in the same city or designing different machines using the same set of pieces. In these cases, the world's dynamics are independent of the task, so, once acquired, the model can be used for solving any new task more efficiently.

One challenging aspect of designing planning methods is that real-world domains are often too complex for simple, tabular representations of the model to be useful. Storing a different value for each possible situation in a game of poker, each geographical location in a city or each possible configuration of a set of pieces is likely to be intractable. Fortunately, the field of machine learning can provide solutions for representing functions over large spaces compactly. These machine learning methods are known as 'function approxi-

mation', and they can be thought of as trading accuracy for compactness and generalization. This thesis will focus on planning methods that represent the model using function approximation.

Another important aspect is that the effects of its actions will often appear to the agent as non-deterministic. For instance, a poker player does not know what the next dealt card will be, a driver cannot predict the precise amount of time that will be spent caught in traffic, and some of the pieces that one tries to build a machine out of might be faulty. Consequently, the planning algorithms introduced in this thesis will use stochastic models to account for this uncertainty.

While it is possible to have the model programmed in by a domain expert, in general one cannot assume that a model is *a priori* available. A simple example is driving in a new city, where knowledge of the traffic patterns is gradually acquired. In such a context, it is important to consider the problem of learning the model from data. For all the approximate model architectures that I propose, I will also discuss methods for learning the parameters of the function approximator.

In this thesis, the objective of the planning process will be to maximize a numeric reward (reinforcement) signal. The reward of a poker playing agent would be the amount of money it makes. For driving, positive rewards for getting to the destination fast must be balanced with large negative rewards for getting into accidents. Learning how to maximize reward from experience is generally known as 'reinforcement learning'. Thus, the methods for planning with learned models investigated in this thesis are also (model-based) reinforcement learning methods.

Another property of many real-world problems is that previous experience may provide useful information that is not present in the agent's current observations. Such systems are usually referred to as 'partially observable'. For instance, poker is partially observable if the playing agent only observes the current table configuration, as important information about an opponent's past behavior is discarded; driving is also partially observable if the driver only uses the current visual observation, because this ignores useful information such as previously observed speed limit signs.

This thesis will not deal with issues related to partial observability explicitly. Rather, I

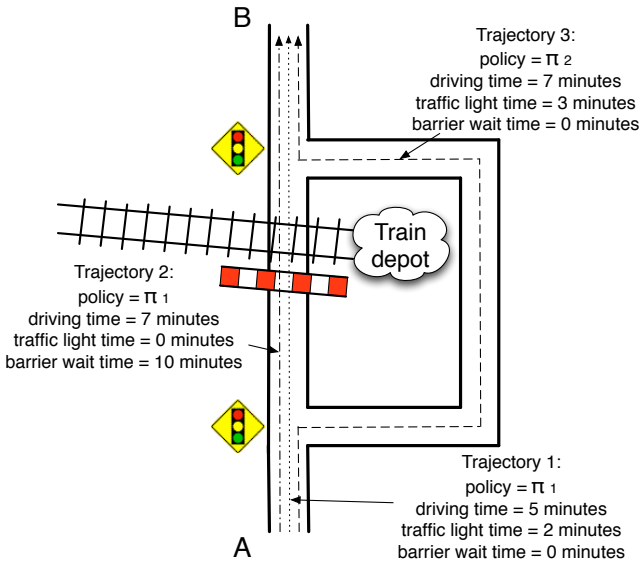

Figure 1.1: Illustration of sampling-based planning.

will make the common assumption that the agent is provided with a representation that contains sufficient information about previous interaction. Under this assumption, the planning problem can be formulated as a Markov decision process (MDP), a standard formalism for describing stochastic decision-making problems.

To summarize, this thesis will investigate the problem of MDP planning with models that are learned, stochastic and represented using function approximation. I will explain how previous methods for addressing this problem are limited and how the methods proposed here remove some of these limitations.

1.1 Sampling-based Planning

The main approach to MDP planning investigated in this thesis uses sampling-based or generative models. As the name indicates, a sampling-based model can be used to sample hypothetical future experience. The key of the planning process is that mechanisms for learning from real experience are used on the sampled data. This idea is not new, dating back at least to Sutton's Dyna work (Sutton, 1990).

To better understand the sampling-based planning process, let us analyze a fragment of the city driving task illustrated in Figure 1.1. The driving agent has to navigate from A to

B, and it can choose between going straight forward over the rail tracks or taking the detour to the right of the figure. The detour is longer and usually requires more waiting at traffic lights, while the straight route has the disadvantage that the barrier might be down if there is an incoming train. In order to choose the route that will minimize the total driving time, the agent has to consider possible sources of non-determinism such as whether the barrier will be lifted or lowered, what color will the traffic lights show or how heavy traffic will be.

A sampling-based model for this problem would generate samples of these events according to their probabilities of occurring. For instance, given that the agent is in front of the barrier and that the probability of the barrier being lowered is 0.2, 'barrier lowered' and 'barrier lifted' would be sampled 20% and 80% of the time, respectively.

The sampling-based model allows the agent to generate possible future trajectories. Given a policy for selecting actions (in this case, a route), these trajectories will be comprised of sampled values for different events that might occur along the way. Three such trajectories are illustrated by the dashed lines in Figure 1.1, two of them generated using policy π_1 (go straight) and one generated using policy π_2 (take the detour). The figure shows sampled waiting times for traffic, barrier or traffic lights.

The sampled trajectories can be used by the agent to update its estimates of the total travel time, which is in turn useful for deciding what route to follow. For instance, if the estimated total time for policy π_2 was initially larger than 10 minutes (the sampled time), this estimate would be decreased after observing Trajectory 3. After updating its estimates, the agent would choose the route (policy) with the smallest estimated waiting time.

An important advantage gained by using sampling-based methods is that the model representation can be completely independent from the policy representation. This allows for great flexibility in the design of a complete learning and planning system: no matter what function approximator is used for representing the model, the function approximator for the policy can be chosen independently so that it best fits the problem. In contrast, other existing approaches to the planning problem (e.g. Bradtke & Barto, 1996; Boutilier et al., 2000; Boyan, 2002; Sallans, 2002; Degris et al., 2006) require particular combinations of model and policy representation.

1.2 Expectation-based Planning

Chapter 6 investigates the possibility of expectation-based planning. Instead of representing full probability distributions over future events, as sampling-based methods would require, expectation-based methods only represent the expected values of those events. For instance, in the driving example an expectation-based model would only predict the average time spent waiting at the barrier for a train to pass.

An important advantage of expectation-based planning over sampling-based planning and other planning methods is that an expectation-based model is easier to represent and learn than a full probabilistic model. This comes, however, at the cost of decreased expressiveness. Going back to the driving example, the same expected barrier waiting time could be caused by either many trains taking a short time each, or by a few trains that take a long time to pass. Being able to distinguish between frequent short waits and infrequent long waits could be important, for instance, if the task is to get from A to B in less than x minutes (e.g. when B is the airport).

1.3 Contributions

Two complete sampling-based planning systems with approximate and learned models, based on existing ideas from statistics and machine learning, are discussed in detail in Chapter 4. The only difference between these two systems is the way the model is represented. The first type of model, called 'independent sampling', ignores the relationships between state variables within the same time step. The second model, called 'cascade sampling', represents these relationships explicitly. In the driving example, independent sampling may produce the pair of events 'barrier lifted' and 'train coming', although the two events never happen together. A cascade sampling model, on the other hand, explicitly encodes the fact that if the barrier is up, then the train cannot be too close.

Unlike previous approximate models used for MDP planning, the cascade sampling architecture does not pose any major constraints on the types of models that can be represented. Its representation power is limited only by the resources available to the function approximator.

Both planning systems are implemented and evaluated empirically in a stochastic, continuous domain, where they are compared to learning directly from experience (model-free learning). For a single task per environment scenario the results indicate that the cascade sampling approach can require less experience to learn a good policy; however, model-free methods perform better than planning systems in the long run. In the case of multiple tasks in the same environment, the advantages of planning methods over model-free methods are shown to be more significant than in the case of a single task. In addition, experiments on randomly generated MDPs are used to illustrate how the quality of the state representation influences the performance of planning methods compared to model-free methods.

Chapter 6 analyzes the conditions under which, despite their limited expressiveness, expectation-based models can still be used as part of a planning system. The results in Chapter 6 are entirely theoretical. These results show the potential of expectation-based planning, but this potential is yet to be incorporated into a complete planning system. Designing and testing such a system requires separate, careful work, and is therefore left for future research.

In addition to discussing particular methods, general theoretical results concerning the performance of planning with approximate models are presented in Chapter 3. These results apply to any planning algorithm, and they bound the quality of the final solution in terms of the quality of the approximate model.

Chapter 2

Background and Related Work

The main goal of this chapter is to describe existing methods for planning and learning with approximate models of Markov decision processes (MDPs). Before discussing these methods, I will provide necessary background on reinforcement learning and MDPs in Sections 2.1 and 2.2.

2.1 Reinforcement Learning

In reinforcement learning an agent interacts with an environment with the purpose of maximizing some function of a numerical reward (reinforcement) signal. The agent can select actions, sense aspects of the environment's state and receive rewards that depend on the actions it selects. This process occurs in a (typically discrete) sequence of time steps and is summarized in Figure 2.1.

Playing poker, driving a car or building a mechanical system, presented in the introduction as examples of planning systems, can also be regarded as instances of the reinforcement learning problem. A poker playing agent observes cards and selects actions such as folding or betting, while trying to maximize the amount of money it leaves the table with. A driver senses other cars or obstacles, perhaps through a vision sensor, and acts by steering or breaking; short travel time would be rewarded, while accidents would be highly penalized. An agent building a mechanical system senses components by seeing or touching them, acts by placing them together and gets rewarded according to the quality of the final product.

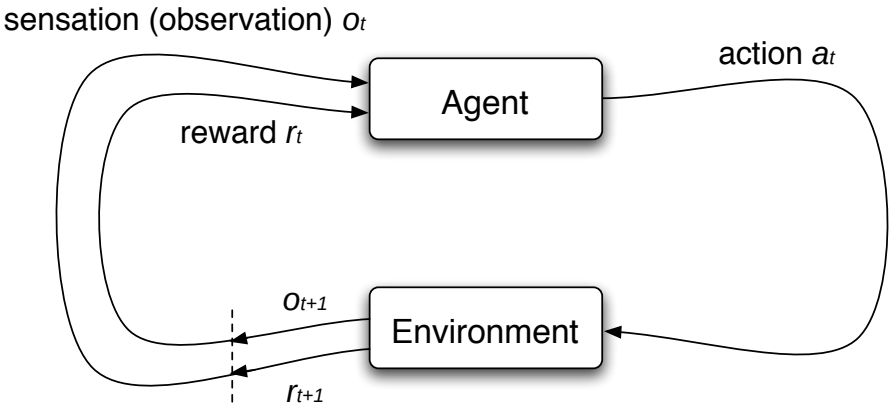

Figure 2.1: Graphical illustration of the agent-environment interaction. At time t, the agent selects action a_t based on the previous observations and rewards, the last of which are o_t and r_t. The environment responds to action a_t by emitting a new observation o_{t+1} and a new reward r_{t+1}.

2.2 Markov Decision Processes

Markov decision processes (MDPs) are a widely used formalism for describing sequential decision making problems. An MDP consists of a tuple $\mathcal{M} =< S, A, P, R >$, where S is a set of states and A is a set of actions. $P : \Sigma \times A \times S \rightarrow [0, 1]$, where Σ is a σ-algebra over S, is a transition function, and $R : S \times A \rightarrow \mathbb{R}$ is a reward function. In this thesis, the action set A will be assumed to be finite. An MDP with $|A| = 1$ is also referred to as a Markov chain.

The transition and reward functions, together with a policy for selecting actions $\pi : S \times A \rightarrow [0, 1]$, can be used to generate sequences $s_0, a_0, r_1, s_1, a_1, ...$ of states, actions and rewards. For each state s and action a, the probability with which the next state is generated is determined by the transition function:

$$Pr(s_{t+1} \in B | s_t = s, a_t = a) = P(B, a, s)$$

where $B \subset S$ is a measurable set. In the particular case of finite state spaces, the transition function simply determines, for each $s' \in S$, $Pr(s_{t+1} = s' | s_t = s, a_t = a)$.

In the most general case, the reward function could have similar semantics. However, since all the methods investigated in this thesis use only the expected value of the next

reward, the reward function will be defined such that:

$$E(r_{t+1}|s_t = s, a_t = a) = R(s, a).$$

Finally, the actions are selected probabilistically according to policy π:

$$Pr(a_t = a|s_t = s) = \pi(s, a).$$

MDPs fit in naturally as models of reinforcement learning, as both are based on the same type of interaction. See the textbooks of Bertsekas and Tsitsiklis (1996) or Sutton and Barto (1998) for an extensive treatment of solutions to the reinforcement learning problem that rely on representing the environment as an MDP.

Notice that an MDP's transition and reward functions solely depend on the current state and action: any information about previous states and actions is irrelevant given the current state. If the agent cannot directly access a complete summary of previous interaction, formulating the reinforcement learning problem as an MDP might be limiting. In this case, models such as partially observable MDPs (POMDPs) (Cassandra, 1994; Littman, 1996) or predictive representations (PSR, TD nets, OOMs) might be more appropriate. In this thesis, however, only the MDP formulation will be investigated.

Besides reinforcement learning problems, planning problems can also be formulated in terms of MDPs. In the MDP formulation of the planning problem, the model is represented by the transition and reward functions, and the objective of the planning process is to compute a policy that maximizes some function of future rewards. For other formulations of the planning problem, one can consult general planning textbooks (e.g. Dean & Wellman, 1991; Ghallab, Nau & Traverso, 2004)

The function of future rewards that has to be maximized depends on whether the problem is formulated as an episodic or continuing task. In the episodic setting, there is a special terminal state. Whenever this terminal state is reached, the current episode ends and a new episode is generated according to π, P and R, starting from some distribution d_0 over the state space. The time step at which the episode terminates is usually denoted by T. In the continuing (or non-episodic) case, there is no terminal state and interaction goes on indefinitely.

A common objective function for the episodic case is the expected sum of future rewards. Under this objective, a policy π maximizing

$$V(\pi) = E_\pi \left[\sum_{t=0}^{T} r_{t+1} \right]$$

must be found. In the previous equation, $V(\pi)$ is called the value of policy π. The policy $\pi^* = \arg\max_\pi V(\pi)$ is called the optimal policy.

In the continuing case, the sum of future rewards is taken over an infinity of terms and can therefore be undefined. In this case, the objective function is the discounted sum of future rewards, defined as

$$V(\pi) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]$$

where $\gamma \in [0, 1)$ is a discount factor that gives exponentially less weight to rewards further on in the future.

Two useful constructs related to this objective function are the state value function (also referred to as the value function) $V^\pi : S \to \mathbb{R}$, defined as

$$V^\pi(s) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^{t+i} r_{i+t+1} | s_t = s \right]$$

and the state-action value function $Q^\pi : S \times A \to \mathbb{R}$, defined as

$$Q^\pi(s, a) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^{t+i} r_{i+t+1} | s_t = s, a_t = a \right].$$

The optimal state value function is then defined as $V^* = V^{\pi_*}$, and the optimal state-action value function as $Q^* = Q^{\pi_*}$. Similar constructs are defined for the episodic case by using $\gamma = 1$ and summing to T instead of ∞ in the above equations.

Another possible objective function for the continuing case is the average of expected future rewards. The function to maximize in this case is

$$V(\pi) = \rho^\pi = \lim_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n} E_\pi \left[r_{i+1} \right].$$

For the average-reward objective, the action-value function is defined as

$$Q^\pi(s, a) = \sum_{i=0}^{\infty} E_\pi \left[r_{t+i+1} - \rho^\pi | s_t = s, a_t = a \right]$$

2.3 Model-free Methods for Reinforcement Learning

Model-free or direct reinforcement learning algorithms learn an estimate of the optimal policy directly from experience, without explicitly building a transition or reward model. Methods that learn a value function as an intermediate step towards learning a policy are often considered to be model-free methods, and they will be presented as such in this thesis.

A variety of algorithms exist for model-free reinforcement learning, most of them based on MDP theory. Comprehensive surveys of such methods can be found elsewhere (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998); here I will only describe, in the following section, several algorithms that are relevant to the rest of this thesis.

2.3.1 Function Approximation in Model-free Reinforcement Learning

When the state space is small enough, the value function or the policy for each state can be represented exactly, for instance in a table. Methods that use such exact representations are usually referred to as 'tabular'. The tabular approach, while conceptually simple, is intractable if the state or action space is large or continuous. The first reason for this is that tabular representations are not compact: the memory requirements of storing a different value for each state-action pair become prohibitive as the size of the state space increases. Second, tabular representations do not lend themselves to generalization: no matter how similar two states are, updating the value of one does not change the value of the other in any way, potentially requiring a large number of updates to be made before anything meaningful can be learned.

Function approximation deals with these issues by trading accuracy for compactness and generalization. A function approximator for function $g : X \rightarrow Y$ is a function $f : X \rightarrow Y$ that can be represented more compactly than g (f typically has a parameterized form, and thus is completely specified by the parameter values). The goal is to find a function f in a certain class of compact functions such that f is as close to g as possible. Surveys of different function approximation techniques can be found in the textbooks of Mitchell (1997) or Hastie, Tibshirani and Friedman (2001).

Function approximation relies on the existence of structure in the world that can be exploited for meaningful generalization. For instance, one of the simplest function approx-

imators simply groups together similar elements of X and stores a separate value for each group. The structural assumption here is that it is reasonable that f outputs the same value for all members of any given group.

An important concept for many approximate reinforcement learning methods is that of feature-based function approximators. It is often the case that the original state representation was designed such that states are easy to interpret by a domain expert, yet this representation might not be an appropriate input for a parametric function approximator. A common solution is to add an extra function, mapping the original state space to a more appropriate space for the function approximator. This additional function will be denoted by $\phi : S \rightarrow \Phi$, where Φ is typically a subset of $\mathbb{R}^n$; the vector $\phi_s = \phi(s)$ will be referred to as the feature vector for state s. More generally, ϕ can be defined over $S \times A$ but, because $|A|$ is assumed to be small, state-action values can be handled by simply using a separate approximator $\hat{Q}^a$ for each action $a \in A$. Thus, the function approximator will be composed of the feature extraction mechanism ϕ and a (typically parametric) function $f : \Phi \rightarrow \mathbb{R}$.

One method for building binary feature vectors is tile coding (e.g. Sutton and Barto, 1998). In tile coding, multiple grids (referred to as tilings) are overlaid on top of the (possibly continuous) state space. The feature vector for state s will contain ones in the positions corresponding to the grid cells that cover s and zeros in all other positions.

Numerous choices of feature-based function approximators have been investigated in the reinforcement learning literature. These include linear, quadratic or other polynomial functions of the features, neural networks (Tesauro, 1995, Lin, 1992) or decision trees (Boutilier et al., 2000; Degris et al., 2006).

A particularly simple, yet powerful form for the value function approximator is one that is linear in the features: $Q_\omega(\phi, a) = \phi^T \omega_a$, where $\omega_a \in \mathbb{R}^n$ is the vector of value function parameters corresponding to action a. Linear approximators have been successfully used in a variety of applications (Sutton, 1996; Stone, Sutton & Kuhlmann, 2005; Sturtevant & White, 2006). They are often preferred because they are easy to analyze and interpret, in addition to having rather low computational costs. Note that, depending on the feature construction mechanism, linear function approximators might be non-linear with respect to the original state space.

Sarsa(λ) (Sutton and Barto, 1998) with linear function approximation is a popular model-free reinforcement learning algorithm. Under linear Sarsa, the following updates are performed at each time step $t > 0$:

$$e := \lambda e + \phi_{s_{t-1}}$$
$$\omega_{a_{t-1}} := \omega_{a_{t-1}} + \alpha \left[r_t + \gamma Q^\pi_\omega(s_t, a_t) - Q^\pi_\omega(s_{t-1}, a_{t-1}) \right] e$$

where ω is the parameter vector of the approximate value function $Q^\pi_\omega(s, a) = \phi_s^T \omega_a$ and e is a real-valued vector called the eligibility vector. The initial value of e is the zero vector, while ω_0 can be arbitrary. The λ parameter takes values in $[0, 1]$ and determines how much the agent bootstraps (learns from other estimated values). The learning rate parameter $\alpha \in (0, 1]$ determines how much influence new experience has on the existing estimate. Model-free reinforcement learning methods that use bootstrapping, such as Sarsa, are usually referred to as 'temporal-difference' (TD) learning methods.

Using Sarsa for control is most commonly done by modifying the policy at every time step so that it is ϵ-greedy with respect to the current value function. An ϵ-greedy policy with respect to value function Q has the form

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = \arg\max_x Q(s, x); \\ \epsilon - \frac{\epsilon}{|A|} & \text{otherwise.} \end{cases}$$

In the average reward case, the R-learning algorithm (Schwartz, 1993) can be used. An on-policy version of R-learning with linear function approximation performs the following updates at every time step:

$$\rho := \rho + \alpha_\rho (r_t - \rho)$$
$$\omega_{a_{t-1}} := \omega_{a_{t-1}} + \alpha [r_t - \rho + Q_\omega(s_t, a_t) - Q_\omega(s_{t-1}, a_{t-1})] \phi_{s_{t-1}}$$

where ρ is an estimate of the average reward, α_ρ and α are learning rate parameters and $Q^\pi_\omega(s, a) = \phi_s^T \omega_a$. Similar to Sarsa, an ϵ-greedy policy with respect to Q_ω can be used for the control case.

2.4 Model-based Methods for Reinforcement Learning

Model-based reinforcement learning involves, as the name suggests, using a model for solving reinforcement learning problems. If the MDP formalism is used for describing the

problem, then model-based reinforcement learning is equivalent to MDP planning.

An important observation is that any model-free reinforcement learning method can be used as part of a model-based method if a sampling-based model of the world is available. As explained in Section 1.1, in this case the agent can plan by applying model-free methods on sampled experience. Specialized methods for planning with sampling-based MDP models also exist (Kearns, Mansour and Ng, 1999; Munos and Szepesvari, 2005).

The value iteration algorithm for MDP planning maintains an estimate of the optimal value function and updates this estimate in a series of iterations. Each new iterate V_{k+1} is computed by $V_{k+1}(s) = BV_k(s)$, where B is the Bellman operator defined, for any value function V, as

$$BV(s) = \max_a \left[R(s,a) + \gamma \int_S P(ds', a, s) V(s') \right], \forall s \in S.$$

For finite state spaces, the integral is replaced by a sum over all $s \in S$. The sequence of value functions produced by value iteration is guaranteed to converge to the optimal value function V^*, under the same conditions for which V^* is guaranteed to exist.

If function approximation is used for the value function, a class of modified variants of the value iteration algorithm generally referred to as 'approximate value iteration' can be used (e.g. de Farias and Van Roy, 2000). The simplest form of approximate value iteration computes V_{k+1} as the best fit to BV_k in the class of value function approximators.

2.4.1 MDP Planning with Approximate Models

For small MDPs, the transition model P can be represented exactly by simply keeping a table that stores for each (s, a, s') tuple the value of $P(s, a, s')$. A similar tabular representation can be used for the expected reward model, storing the value of $R(s, a)$ for each (s, a) pair. The table-based approach is, however, infeasible for large or continuous state spaces. In such cases, function approximation has to be used for representing the model.

As discussed previously, the premise that justifies the use of function approximation is that the world has structure, enabling a compact, approximate model to be learned and used. Correspondingly, existing methods for approximating the model make different assumptions about the type of structure that the world has.

A simple solution is to discretize the state space into a small number of aggregate states and use a table to store transition probabilities between pairs of aggregate states (Kuvayev & Sutton, 1996). This approach suffers from the general problem of state-aggregation methods: an increase in the discrimination power can only be achieved at the cost of decreased generalization, and vice versa.

Another approach is to learn a model of the world as a factored dynamic Bayes network (DBN). If state variables with conditional independence relationships can be identified, then using DBNs for learning and representing an MDP model is simple and efficient. If such conditional independence relationships are not present in the system, however, then DBNs alone do not offer any advantages over tabular representations.

When the conditional independence relationships exist and can be identified, models of MDPs with a large number of states can be represented exactly using a small number of parameters. Correspondingly, DBN transition models using tabular representations of the dependencies between state variables have previously been used for MDP planning (Tadepalli & Ok, 1996; Andre, Friedman & Parr, 1998; Guestrin et al., 2003). Guestrin, Hauskrecht and Kveton (2004) use a similar approach for representing distributions over continuous state variables, using a table to store the parameters of conditional distributions from the exponential family. When the conditional independence assumptions are not met, the number of parameters that these methods require is exponential in the number of state variables.

Boutilier, Dearden and Goldszmidt (2000) use a representation of the DBN in the form of decision trees, allowing for larger state spaces to be represented compactly. For planning with this model they propose a particular algorithm called structured value iteration (SVI) that takes advantage of the tree structure. Degris, Sigaud and Wuillemin (2006) build on the work of Boutilier et al. by extending it to the case of learned models.

Sallans (2002) proposed a more general representation, where the DBN model of $P(s', a, s)$ only assumes conditional independence between the components of s'. In the case of multidimensional binary state representations, the function approximator for representing the probability of each component $s'(i)$ of s' was a log-linear combination of the

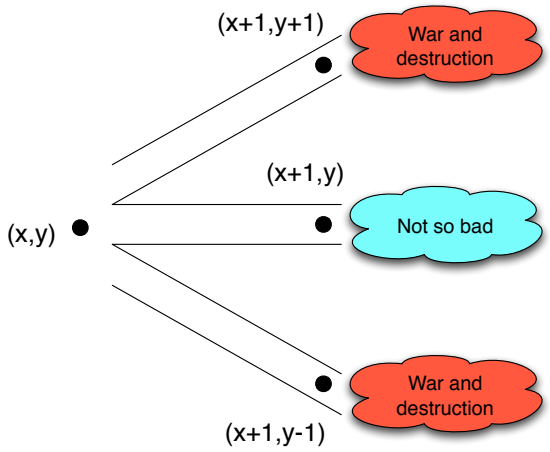

Figure 2.2: Illustration of the possible pitfalls of only considering the expected next state.

components of s:

$$Pr(s'(i) = 1|s, a) = \sigma\left(s^T \Theta^a(i)\right)$$

where $\Theta^a(i)$ is the portion of the model parameters Θ corresponding to action a and component i, and σ is the logistic function $\sigma(x) = 1/(1 + e^{-x})$. In order to use this model in a planning system, Sallans makes additional assumptions about the structure of the value function.

Another simplifying assumption is that the transition distribution is of a special parametric form, most commonly Gaussian. This is the case, for instance, in some robotics applications (Ng et al, 2004; Kaboli, Bowling & Musilek, 2006).

Other researchers have investigated planning with approximate models of deterministic systems (e.g. Atkeson, 1993; Atkeson, Moore & Schaal, 1997; Nouri & Littman, 2006). Determinism makes it easy to represent the model: instead of a full probability distribution, only one value needs to be predicted for the next state. Their methods have been successfully applied on a number of challenging problems, including domains with continuous state spaces. If the real world is stochastic, however, ignoring this stochasticity can be unsound, as illustrated by the following example.

For illustration purposes, consider the domain in Figure 2.2. Assume that after moving forward from (x, y) the agent ends up in either $(x+1, y)$, $(x+1, y+1)$ or $(x+1, y-1)$, each of them with equal probability, and that no reward is received for any of these transitions.

When applied directly on this domain, the methods mentioned in the previous paragraph would learn a model that predicts the expectation of the next state distribution given the current state and an action. Using such a model with TD methods and a Dyna-style planning system is hazardous. The expectation of the next state distribution after moving forward from (x, y) is $(x+1, y)$ and, since this is the only information provided by the model, a TD method using this model would always update the value of state (x, y) towards the value of $(x + 1, y)$. Because this ignores the other possible future states, the agent never learns that by going forward from (x, y) it might also end up in $(x+1, y+1)$ or $(x+1, y-1)$ and from that to 'war and destruction'. Thus, the value of (x, y) always gets updated based on the value of $(x+1, y)$ and never based on the value of $(x+1, y+1)$ or $(x+1, y-1)$ (presumably the imminence of war and destruction would cause the value function for $(x+1, y+1)$ and $(x + 1, y - 1)$ to be very, very low).

The example above does not imply that a deterministic, expectation-based model can never be used, only that we have to be careful about how to use it. In fact, in Chapter 6 I will show that under certain conditions there are reasons to believe that planning with expectation-based models can be performed in a sound manner.

Least-squares temporal difference (LSTD) methods (Bradtke & Barto, 1996; Boyan, 2002; Geramifard, Bowling & Sutton, 2006) can also be considered model-based methods, although they do not explicitly build a transition model. Instead, a model of how feature vectors at consecutive time steps are correlated is built, and this model is used to perform policy evaluation. LSTD methods rely on the value function approximator being linear.

Because the model that LSTD builds is policy-dependent, in the control setting a new model has to be learned every time the policy changes. To limit the amount of experience that this process requires, Lagoudakis and Parr (2003) and Antos, Szepesvari and Munos (2006) propose algorithms that use a single stored trajectory to build an LSTD-style model for any policy. Lagoudakis and Parr offer no performance guarantees concerning their 'least-squares policy iteration' (LSPI) algorithm, whereas Antos, Szepesvari and Munos provide finite-sample bounds for the performance of their 'Bellman-residual minimization based fitted policy iteration' method.

2.4.2 Learning Models for MDP Planning

This section will survey existing methods for planning with learned models. The emphasis will be on learning transition models; learning an expected reward model is an easier problem, since it avoids dealing with a full probability distribution.

Depending on how the learned model is used, two types of approaches are common in the literature. The first is comprised of a separate model learning phase and a separate planning phase. In the model learning phase, the model is learned from data using a random or hand-coded control policy. After the model is learned, it is fixed and a plan is computed off-line using the learned model (e.g. Ng et al., 2004).

The second class of methods interleave planning and learning, and is illustrated by the Dyna architecture (Sutton, 1990; Singh, 1992; Degris, Sigaud & Wuillemin, 2006). A model of the world is learned from experience, and at the same time a decision-making policy is computed based on the learned model. Extensions such as prioritized sweeping (Moore & Atkeson, 1993) and Queue-Dyna (Peng & Williams, 1993) improve computational efficiency by guiding the planning process from goals or states that have recently changed value.

As in the value function case, learning the transition model depends on the way the model is represented. If a table-based approach is used, then learning can simply proceed by frequency counts (e.g. Kuvayev & Sutton, 1996). Frequency counts can also be used for the table-based DBN representations mentioned in section 2.4.1 (Tadepalli & Ok, 1996; Andre, Friedman & Parr, 1998; Guestrin et al., 2003).

More interesting is the recent work of Degris, Sigaud and Wuillemin (2006), in which a decision tree DBN representation is learned from data. Thus, less experience is required because of the generalization induced by the decision tree. Learning an approximate DBN model has also been investigated using mean field theory (Sallans, 2002).

For planning methods that use deterministic models (Atkeson, 1993; Atkeson, Moore & Schaal, 1997; Nouri & Littman, 2006) the model-learning problem amounts to traditional supervised learning, and a variety of methods are available.

In the cases when the transition distribution is represented in a parametric form, the

learning problem is easy if a set of parameters for each state can be stored in a table. However, if function approximation has to be used for the state-dependent distribution parameters, learning can be more complicated. The main problem is getting training data of the form (s, θ), where θ is the set of transition model parameters corresponding to state s.

Learning the LSTD model is straightforward, as it amounts to estimating a correlation matrix between the feature vector components.

Many of the learning and planning algorithms mentioned above have been empirically shown to achieve better data-efficiency than model-free methods.

2.5 Conclusion

Existing methods for learning and planning with approximate MDP models were surveyed in Sections 2.4.1 and 2.4.2. While having obvious merits, these methods are limited in the class of systems they can model by the structural assumptions that they make. The methods investigated in this thesis aim to alleviate some of these restrictions.

Chapter 3

The Effect of Model Inaccuracies

A fundamental question for planning with approximate models is the following: If a planning method computes a policy using an imperfect world model, how well will that policy perform in the real world? The theoretical results presented in this chapter guarantee that, if the approximate model is accurate enough and an appropriate planning method is used, the real-world performance will be close to optimal.[1]

3.1 Theoretical Results

The following results (Theorem 1 and Theorem 2) bound the difference between the optimal value functions of two MDPs in terms of the distance between their models. The theorems analyze the (discounted or undiscounted) cumulative return case; the average reward formulation is not discussed here.

The analysis uses L^p norms, defined as $\|f\|_p = \left(\int |f(x)|^p dx\right)^{1/p}$. The limit as p goes to ∞ of the L^p norm is called L^∞, defined as $\|f\|_\infty = \sup_x |f(x)|$. The L^1 norm will be used to measure the distance between the transition models in the text of Theorems 1 and 2, as it is considered to be a "natural distance" for measuring the distance between two probability distributions (Devroye and Györfi, 1985).

The first result can now be stated:

Theorem 1. *Given two MDPs $\mathcal{M}_1 = (S, A, P_1, R_1)$ and $\mathcal{M}_2 = (S, A, P_2, R_2)$ with the same discount factor $\gamma < 1$, the distance between the two optimal value functions V_1^* and*

[1] The results presented in this chapter are the outcome of joint work with Csaba Szepesvari.

V_2^* can be bounded by

$$\|V_1^* - V_2^*\|_\infty \leq \frac{1}{1-\gamma}\left(\|R_1 - R_2\|_\infty + \gamma\|V_2^*\|_\infty \max_a \sup_s \|P_1(\cdot, a, s) - P_2(\cdot, a, s)\|_1\right).$$

Proof. Using the triangle inequality,

$$\|V_1^* - V_2^*\|_\infty = \|B_1V_1^* - B_2V_2^*\|_\infty \leq \|B_1V_1^* - B_1V_2^*\|_\infty + \|B_1V_2^* - B_2V_2^*\|_\infty. \quad (3.1)$$

It is well known that the Bellman operator is a contraction with contraction factor γ. Thus, the first term can be bounded by:

$$\|B_1V_1^* - B_1V_2^*\|_\infty \leq \gamma\|V_1^* - V_2^*\|_\infty \quad (3.2)$$

For the second term

$$\begin{aligned}
\|B_1V_2^* - B_2V_2^*\|_\infty &\leq \sup_s |\max_a [R_1(s,a) - R_2(s,a)]| \\
&\quad + \gamma \sup_s \left|\max_a \left[\int V_2^*(y)P_1(dy,a,s) - \int V_2^*(y)P_2(dy,a,s)\right]\right| \\
&\leq \sup_s \max_a |R_1(s,a) - R_2(s,a)| \\
&\quad + \gamma \sup_s \max_a \left|\int (V_2^*(y)P_1(dy,a,s) - V_2^*(y)P_2(dy,a,s))\right| \\
&\leq \|R_1 - R_2\|_\infty \\
&\quad + \gamma \sup_s \max_a \int |V_2^*(y)\,(P_1(dy,a,s) - P_2(dy,a,s))| \\
&\leq \|R_1 - R_2\|_\infty \\
&\quad + \gamma \sup_s \max_a \|V_2^*\|_\infty \int |P_1(dy,a,s) - P_2(dy,a,s)| \\
&= \|R_1 - R_2\|_\infty \\
&\quad + \gamma \sup_s \max_a \|V_2^*\|_\infty \|P_1(dy,a,s) - P_2(dy,a,s)\|_1
\end{aligned}$$

The theorem's statement can now be obtained by replacing the two terms in the right hand side of Equation 3.1 with their respective bounds. □

For the episodic, undiscounted case, a contraction property of the Bellman operator in a weighted supremum norm can be used instead of Equation 3.2. Given $f : X \to \mathbb{R}$ and $w : X \to \mathbb{R}$ such that $w(x) > 0, \forall x \in X$, the weighted supremum norm of f with respect to w is defined as $\|f\|_{w,\infty} = \sup_x |f(x)|/w(x)$. The following result can now be stated:

Theorem 2. *Given two MDPs $\mathcal{M}_1 = (S, A, P_1, R_1)$ and $\mathcal{M}_2 = (S, A, P_2, R_2)$, and given that all policies that could be defined for $\mathcal{M}_1$ or $\mathcal{M}_2$ are proper (there exists $n > 0$ such that there is a positive probability that the terminal state will be reached in at most n steps using any policy from any starting state), the distance between the optimal value functions V_1^* and V_2^* of the two MDPs can be bounded by*

$$\|V_1^* - V_2^*\|_{\xi,\infty} \leq \frac{1}{1-c}\left(\|R_1 - R_2\|_{\xi,\infty} + \|V_2^*\|_{\xi,\infty} \max_a \sup_s \|P_1(\cdot, a, s) - P_2(\cdot, a, s)\|_1\right)$$

where, for any $s \in S$, $\xi(s)$ is the expected number of time steps the current policy would take to reach a terminal state when starting from s, and $c = \sup_{s \in S} 1 - \frac{1}{\xi(s)}$.

Proof. In the undiscounted episodic case, the Bellman operator is a contraction in the $\|\cdot\|_{\xi,\infty}$ norm with contraction factor c if all policies are proper (Bertsekas and Tsitsiklis, 1996, page 23). Using this contraction property instead of Equation 3.2, the bound in the text of the theorem can be proven in a similar way with Theorem 1. □

3.2 Discussion

Theorems 1 and 2 do not mention approximate models explicitly, but they are nevertheless relevant to planning with approximate models. Let $\mathcal{M} = (S, A, P, R)$ be the MDP of interest. A method for planning with an approximate model $(\hat{P}, \hat{R})$ of $\mathcal{M}$ can be thought of as a method for planning with the exact model of $\hat{\mathcal{M}} = (S, A, \hat{P}, \hat{R})$. Theorems 1 and 2 can then be used to bound the difference between the optimal value functions of $\mathcal{M}$ and $\hat{\mathcal{M}}$ based on how close the approximate model is to the correct model.

Using the idea in the above paragraph, one can combine bounds on the performance of a particular planning method and of a particular model-learning technique into a result describing the performance of the integrated planning and learning system. Assume that planning method `plan` is able to compute for any discounted MDP $\hat{\mathcal{M}}$ a policy $\hat{\pi}$ for which $\|V^{\hat{\pi}} - V^*_{\hat{\mathcal{M}}}\|_\infty \leq d$, for some $d \geq 0$. Also assume that model-acquisition method `acquire-model` can compute for any MDP $\mathcal{M} = (S, A, P, R)$ an approximate model $(\hat{P}, \hat{R})$ such that $\|R - \hat{R}\|_\infty \leq \delta_R$ and $\max_a \sup_s \|P(\cdot, a, s) - \hat{P}(\cdot, a, s)\|_1 \leq \delta_P$ for some $\delta_R, \delta_P > 0$. Then a system using the `plan` method with the model computed by

`acquire-model` will be able to compute a policy π such that

$$\|V^{\pi} - V^{*}_{\mathcal{M}}\|_{\infty} \leq d + \frac{\delta_R + \gamma \|V^{*}_{\mathcal{M}}\|_{\infty} \delta_P}{1 - \gamma}$$

Incorporating existing results about the performance of planning and model-learning methods into such a bound is a subject of future work.

Finally, it should be mentioned that results similar to Theorem 1 exist in the literature (e.g. Kalmar, Szepesvari and Lorincz, 1998; Kearns and Singh, 2002; Kakade, Kearns and Langford, 2003). These results hold for MDPs with finite state spaces, but the ideas of their proofs might be easily extended to continuous state spaces – the proof of Theorem 1 is, in fact, based on the similar result of Kalmar, Szepesvari and Lorincz (1998).

Chapter 4

Two Methods for Sampling-based Planning

As discussed in Section 2.4, sampling-based methods provide a general mechanism for MDP planning. The current chapter proposes two methods for sampling-based MDP planning with learned and approximate models. The difference between the two methods is in the way the generative model is represented. The method called iplanner, based on an independent sampling model, is conceptually simple but rather limited, as it ignores dependencies between different components of the feature vector. On the other hand, the cplanner method (based on a cascade sampling model) represents these dependencies explicitly; it will be argued in the Conclusion of this chapter that cplanner is less restrictive than existing methods for planning and learning with approximate MDP models.

4.1 The Sampling-based Planning Process

The class of sampling-based planning methods described here apply direct reinforcement learning algorithms on trajectories sampled from a generative model. For feature-based function approximators, the trajectories are sequences of feature vectors, actions and rewards rather than states, actions and rewards.

A generic algorithm for sampling-based planning is outlined in Table 4.1. This algorithm is meant to illustrate the flow of the process; the functions called by the generic algorithm can have an arbitrary form as long as they behave as described below.

Using the `model-learn` function, the agent learns a (possibly approximate) model of its environment from the transitions observed transitions of the form (ϕ, a, r, ϕ'). After

```
learn-and-plan(φ, π, c1, c2, N)
    a ← sample-action(φ, π)
    repeat
        Take action a, observe r and φ'
        model-learn(φ, a, r, φ')
        a' ← sample-action(φ', π)
        π ← policy-update(φ, a, r, φ', a')
        for i = 1 : N
            Select initial feature vector φ̄ and action ā for new traejctory
            repeat
                r̄ ← expected-reward(φ̄, ā)
                φ̄' ← sample-next-features(φ̄, ā)
                ā ← sample-action(φ̄', π)
                π ← policy-update(φ̄, ā, r̄, φ̄', ā')
                φ̄ ← φ̄', ā ← ā'
            until c2
        end
        φ ← φ', a ← a'
    until c1
```

Table 4.1: Generic algorithm for sampling-based planning with a learned model.

each step of interaction, a variable number of planning steps can be taken, in which data is generated from the learned model by using the `expected-reward` and `sample-next-features` functions. The sampled data is used by `policy-update` to improve the current policy π, perhaps via updating a value function estimate (`policy-update` can also be used with real experience). All actions are generated by `sample-action` according to the current policy. Particular choices for all these functions will be presented in the next sections.

Notice that data is sampled from the model in two nested loops. The outer loop ensures that N independent trajectories will be sampled. The inner loop samples each of the N trajectories. Each trajectory ends when condition c_2 is satisfied. In the particular implementation described in Section 4.7, all trajectories start from the current feature vector, and c_2 is satisfied either if the episode's termination is sampled or if a trajectory of length $L = 20$ has been generated. If c_2 is such that the length of a sampled trajectory is always zero, the algorithm in Table 4.1 becomes generic model-free reinforcement learning.

Also note that, by appropriately choosing N and c_2, this generic algorithm can be regarded as either an on-line or an off-line method. For instance, if the agent learns from

simulated data after every step of real interaction we obtain an on-line, Dyna-style algorithm. In another scenario, real experience could be used only for learning the model until, say, t time steps are elapsed. At time t the agent would go into an off-line planning mode: it would sample many long trajectories from the learned model and compute a policy from these trajectories.

4.2 Updating the Policy

Any reinforcement learning algorithm can be used for the `policy-update` function. It can be Q-learning, policy gradient, LSPI, direct policy search, or any other method that can update the policy given transitions of the form (ϕ, a, r, ϕ', a'). Independent of the model's representation, any function approximator can be used for representing the policy (and the value function, if a value function is used) as long as it uses the same features.

The particular system described in Section 4.7 uses the Sarsa(0) algorithm for updating the parameters of the linear value-function approximator, in conjunction with keeping the policy ϵ-greedy with respect to the current value function estimate. The average-reward experiments in Section 5.4 use R-learning with linear function approximation.

4.3 Generative Models of Multivariate Distributions

The `sample-next-features` function in the generic planning algorithm will use a generative model of the MDP. A feature-based generative model must provide a mechanism that, given a feature vector ϕ and an action a, allows for a feature vector $\bar{\phi}_{t+1}$ to be sampled from $P(\phi, a, \cdot) = Pr(\phi_{t+1} = \cdot | \phi_t = \phi, a_t = a)$, where ϕ_t and a_t are the feature vector and action time t. An important challenge here is to devise such a mechanism for the case of multidimensional feature vectors. In this section I will present two approaches that reduce sampling an n-dimensional feature vector to sampling n univariate random variables.

4.3.1 Independent Sampling

A simple mechanism for sampling multidimensional feature vectors assumes that the components of ϕ_{t+1} are independent given ϕ_t and a_t. Under this assumption, one only needs to

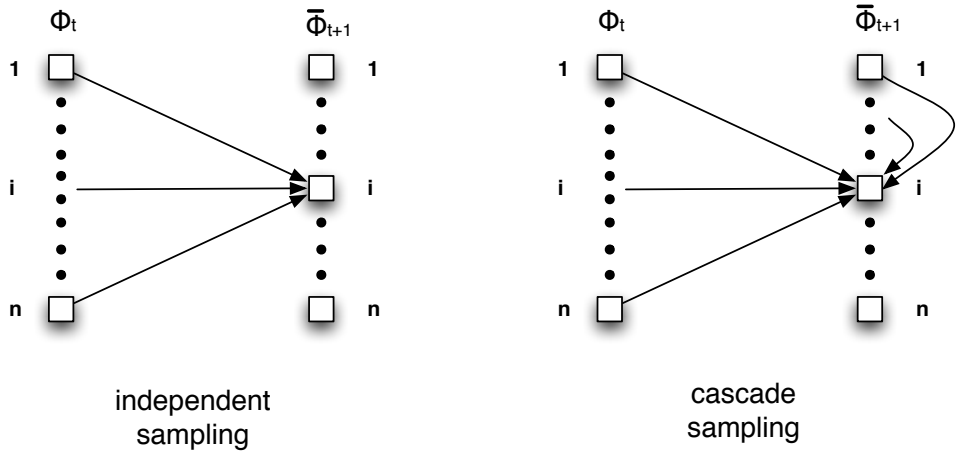

Figure 4.1: Graphical illustration of how component i of $\bar{\phi}_{t+1}$ is generated under independent and cascade sampling, respectively.

sample each component $\bar{\phi}_{t+1}(i)$ of $\bar{\phi}_{t+1}$ from the distribution

$$Pr(\phi_{t+1}(i) = \cdot | \phi_t = \phi, a_t = a).$$

As a side note, this assumption is common in the literature on DBN models.

This previous approach, while conceptually and computationally simple, can be guaranteed to work only when the components of ϕ_{t+1} are statistically independent given ϕ_t and a_t. Otherwise, aspects of the joint probability distribution, such as correlations between different components, might need to be modeled.

4.3.2 Cascade Sampling

The cascade sampling architecture (name inspired from Leslie Kaelbing's thesis (Kaelbling, 1993)) described in this section can model all aspects of the joint distribution given appropriate conditional models of univariate random variables. The main idea behind this architecture is that features are generated in a sequence, and, as the generative process advances through this sequence, the probabilities with which new features are sampled depend on the sampled values of the previous features.

More formally, the generative process for sampling $\bar{\phi}_{t+1}$ from $Pr(\phi_{t+1} = \cdot | \phi_t = \phi, a_t = a)$ begins by sampling the first component, $\bar{\phi}_{t+1}(1)$. This is done, as in the independent architecture, by sampling from $Pr(\phi_{t+1}(1) = \cdot | \phi_t = \phi, a_t = a)$. The second

component, however, depends on the already sampled value of the first component, as it is sampled from $Pr(\phi_{t+1}(2) = \cdot|\phi_t = \phi, a_t = a, \phi_{t+1}(1) = \bar{\phi}_{t+1}(1))$. In general, the i-th component $\bar{\phi}_{t+1}(i)$ is sampled from

$$Pr\left(\phi_{t+1}(i) = \cdot|\phi_t = \phi, a_t = a, \phi_{t+1}(1:i-1) = \bar{\phi}_{t+1}(1:i-1)\right),$$

where $\phi(1 : i)$ denotes the first i components of vector ϕ. Thus, the value of $\bar{\phi}_{t+1}(i)$ depends on all the previously sampled $i-1$ components. Figure 4.1 illustrates this process, contrasting it to independent sampling.

Using this mechanism, sampling from the multivariate next feature distribution is once again reduced to sampling from n univariate conditional distributions, with the added advantage that the new process explicitly models dependencies between features. In fact, by using the sampling process described above, the resulting vector $\bar{\phi}_{t+1}$ will be drawn from the desired distribution $Pr\left(\phi_{t+1} = \cdot|\phi_t = \phi, a_t = a\right)$ regardless of the lack of independence between state features. This can be immediately inferred from the following well-known formula:

$$Pr(X(1) = x_1, X(2) = x_2, ...) = \prod_i Pr(X(i) = x_i|X(1) = x_1, ...X(i-1) = x_{i-1})$$

by making the probabilities conditional on the previous action and feature vector.

Because of this, cascade sampling is significantly more general than independent sampling. Independent sampling might be unable to generate the correct joint distribution even if it was allowed to use arbitrarily complex function approximation for the univariate distributions. Cascade sampling, on the other hand, can achieve an arbitrarily close approximation to the true distribution, whatever it may be, as the function approximators are made more complex and powerful. A potential drawback of cascade sampling, however, is that errors in approximating the distribution of the first components can propagate, thus affecting the quality of approximation for subsequent components.

Similar architectures have previously been used to model relationships between multi-dimensional representations, such as in the work by Bengio and Bengio (2000) on building unconditional models of univariate distributions or Kaelbling (1993) on assigning credit to binary features in reinforcement learning problems.

4.4 Approximating the Univariate Distributions

The types of architecture described above reduce the problem of modeling a conditional distribution over multidimensional features to modeling n univariate distributions. Under these architectures, each of the univariate distributions is allowed to have arbitrary form. However, exactly representing arbitrary distributions over real-valued random variables would require a potentially infinite number of parameters (or stored data points, for non-parametric methods). For a compact representation to be possible, one has to make assumptions either about the form of the distribution or about the range of the values that the random variable can take.

The assumption made in this work is that the components of the feature vectors are binary. Note, however, that general discrete-valued feature vectors can be transformed to binary vectors by using an indicator function for each possible value of each feature. This will increase the feature vectors' size, and is a feasible approach when the features have a relatively small number of possible values.

An important advantage of working with binary features is that each of the univariate distributions can be represented compactly. Indeed, to represent any probability distribution over a binary random variable $X \in \{0, 1\}$ one only has to define $Pr(X = 0)$ and $Pr(X = 1)$. In fact, since $Pr(X = 0) = 1 - Pr(X = 1)$, just one of the two values is sufficient. Thus, all that is needed for sampling $\bar{\phi}_{t+1} \sim Pr\left(\phi_{t+1} = \cdot | \phi_t = \phi, a_t = a\right)$ are the probabilities

$$Pr\left(\phi_{t+1}(i) = 1 | \phi_t = \phi, a_t = a\right)$$

for independent sampling, or

$$Pr\left(\phi_{t+1}(i) = 1 | \phi_t = \phi, a_t = a, \phi_{t+1}(1 : i-1) = \bar{\phi}_{t+1}(1 : i-1)\right)$$

for cascade sampling, where $i = 1, ..n$.

Representing these probabilities in a table would generally be infeasible, since the size of the table would be exponential in the dimensionality of the features space, so function approximation must be used. Any approximation mechanism that models functions $f : \{0, 1\}^n \rightarrow [0, 1]$ can be used here. A popular choice, and the one that will be used in

the experiments in this thesis, is a log-linear combination of the features $f(\phi) \approx \sigma(\phi^T \theta)$, where $\sigma : \mathbb{R} \rightarrow [0, 1]$ is the logistic or sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. Thus, depending on what type of architecture is used, each of the univariate distributions will be approximated as

$$Pr\left(\phi_{t+1}(i) = 1 | \phi_t = \phi, a_t = a\right) \approx \sigma\left(\phi_t^T \theta_a^i\right) \tag{4.1}$$

for the independent sampling architecture, or

$$\begin{aligned} Pr\left(\phi_{t+1}(i) = 1 | \phi_t = \phi, a_t = a, \phi_{t+1}(1:i-1) = \bar{\phi}_{t+1}(1:i-1)\right) \\ \approx \sigma\left(\left[\phi_t^T, \bar{\phi}_{t+1}(1:i-1)^T\right] \theta_a^i\right) \end{aligned} \tag{4.2}$$

for the cascade sampling architecture. Here, for each $a \in A$ and $i \in 1, ..n$, θ_a^i is the parameter vector corresponding to action a and component $\phi_{t+1}(i)$. The use of the logistic function ensures that the output of this approximation scheme will always be between zero and one.

An important question is whether the restriction to binary features is sensible from a practical perspective. The answer seems to be yes, since many non-trivial reinforcement learning problems have been successfully tackled using binary feature vectors (Sutton, 1996; Stone, Sutton & Kuhlmann, 2005; Sturtevant & White, 2006). This is largely due to the fact that methods such as tile coding are capable of turning continuous states into binary feature vectors, in a manner that is useful for both discrimination and generalization.

4.5 Learning the Generative Model

The previous sections described architectures for approximating the model. The current section describes methods for learning the parameters of these architectures from data. In other words, it presents a possible implementation of the `model-learn` function that updates the parameters of the model approximator given a transition of the form (ϕ, a, r, ϕ'). The form of the `model-learn` function will obviously depend on the choice of approximate representation for the model. Still, given a particular approximation architecture, any learning method that is appropriate for that architecture can be used.

The particular implementation proposed here learns a separate model for each of the univariate component distributions appearing in independent or cascade sampling. The

learning method used for each of these components is an on-line version of logistic regression (e.g. Jordan, 1996), a popular choice for log-linear representations. The objective of logistic regression to find the parameters of an approximate distribution model, such that the logarithm of the likelihood that the observed data was generated by that model is maximized. On-line gradient descent on this objective function yields the update rule

$$\theta_a^i := \theta_a^i + \beta \left(\phi'(i) - \sigma\left(\phi^T \theta_a^i\right)\right) \phi$$

for the independent sampling architecture, or

$$\theta_a^i := \theta_a^i + \beta \left(\phi'(i) - \sigma\left(\left[\phi^T, \phi'(1:i-1)^T\right] \theta_a^i\right)\right) \left[\phi^T, \phi'(1:i-1)^T\right]^T$$

for the cascade sampling architecture. Here, $\beta \in \mathbb{R}^+$ is a step-size parameter. Note that, when learning the model parameters for cascade sampling, the values of $\phi'(1:i-1)$ are observed and do not need to be sampled.

4.6 Approximating and Learning the Reward Model

Designing a function approximator for the reward model is relatively simple since, for each feature vector ϕ and action a, the approximator for $R(\phi, a)$ only needs to produce a single number (the expected reward) rather than a full distribution.

The particular approximator used in this thesis is linear in the features, $R(\phi, a) \approx \phi^T \theta_a^R$, where θ_a^R is a portion of the reward model parameter θ^R (which is in turn a portion of the overall model parameter Θ). For learning the model parameters, on-line gradient descent on the mean squared error between the observed rewards and the predicted rewards yields the update

$$\theta_a^R := \theta_a^R + \beta^R \left(r - \phi^T \theta_a^R\right) \phi,$$

where $\beta^R \in \mathbb{R}^+$ is the step-size parameter for the reward model.

With respect to the generic planning algorithm in Table 4.1, reward learning would be part of the `model-learn` function. The learned model would then be used by the `expected-reward` function to produce the next expected reward.

`cplanner`(ω, c_1, N, L)

$a \leftarrow \epsilon$-`greedy`(ϕ, Q_ω)

repeat

Take action a, observe r and ϕ'

for $i = 1 : n$

$$\theta_a^i \leftarrow \theta_a^i + \beta \left(\phi'(i) - \sigma \left(\left[\phi^T, \phi'(1:i-1)^T \right] \theta_a^i \right) \right) \left[\phi^T, \phi'(1:i-1)^T \right]^T$$

end

$\theta_a^R \leftarrow \theta_a^R + \beta^R \left(r - \phi^T \theta_a^R \right) \phi$

$a' \leftarrow \epsilon$-`greedy`(ϕ', Q_ω)

$\omega_a \leftarrow \omega_a + \alpha \left[r + \gamma Q_\omega(\phi', a') - Q_\omega(\phi, a) \right]$

$\bar{\phi} \leftarrow \phi'; \bar{a} \leftarrow a'$

for $j = 1 : N$

$l \leftarrow 0$

repeat

$\bar{r} \leftarrow \phi^T \theta_{\bar{a}}^R$

for i = 1:n

Sample $\bar{\phi}'(i) \approx \sigma \left(\left[\bar{\phi}^T, \bar{\phi}'(1:i-1)^T \right] \theta_{\bar{a}}^i \right)$

end

$\bar{a}' \leftarrow \epsilon$-`greedy`($\bar{\phi}', Q_\omega$)

$\omega_{\bar{a}} \leftarrow \omega_{\bar{a}} + \alpha \left[\bar{r} + \gamma Q_\omega(\bar{\phi}', \bar{a}') - Q_\omega(\bar{\phi}, \bar{a}) \right]$

$\bar{\phi} \leftarrow \bar{\phi}', \bar{a} \leftarrow \bar{a}', l \leftarrow l + 1$

until $l < L$ or end of episode is sampled

end

$\phi \leftarrow \phi', a \leftarrow a'$

until c_1

Table 4.2: Planning and learning with a cascade sampling model (`cplanner`). $Q_\omega(\phi, a) = \omega_a^T \phi$ is the linear action-value-function approximator. The corresponding version for independent sampling is called `iplanner`.

4.7 A Complete Implementation

Table 4.2 describes a complete learning and planning system using cascade sampling, linear reward models and linear Sarsa(0). This algorithm will be called *cplanner*. If independent sampling is used instead of cascade sampling, the corresponding algorithm will be called *iplanner*.

4.8 Computational Complexity

The main motivation for model-based reinforcement learning is that in many practical problems computation is relatively cheap but experience is expensive, and thus it is sensible to allocate more computational resources in order to obtain better data efficiency. However, computational resources are not unlimited, and it is important to know how much time and memory model-based methods will require.

Both iplanner and cplanner require $O(n^2|A|)$ memory. For both methods, the reward part of the model, Θ^R, requires exactly $n \times |A|$ values to be stored. The exact size of the transition model, on the other hand, depends on which of the methods is used: iplanner needs to store $|A|n$ parameter vectors of length n each for a total of $|A|n^2$ elements, while the $|A|n$ vectors that cplanner needs to store are of length $n, n+1, ..., n+n-1$ for a total of $|A|(n^2 + n(n-1)/2)$ elements.

A similar argument can be used to show that, for both cascade and independent sampling with log-linear models, the time complexity of sampling the next feature vector ϕ' given the current vector ϕ and an action a is $O(k(n + |A|))$, where k is the number of non-zero elements of ϕ. For the complete learning and planning system described in Table 4.2, the total time complexity per step of real interaction is $O(NLk(n + |A|))$.

In comparison, model-free reinforcement learning algorithms with linear value function approximation such as linear Sarsa(0) or linear Q-learning have very low computational complexity. They only require $O(n|A|)$ memory and $O(k|A|)$ computation for every step of interaction with the environment.

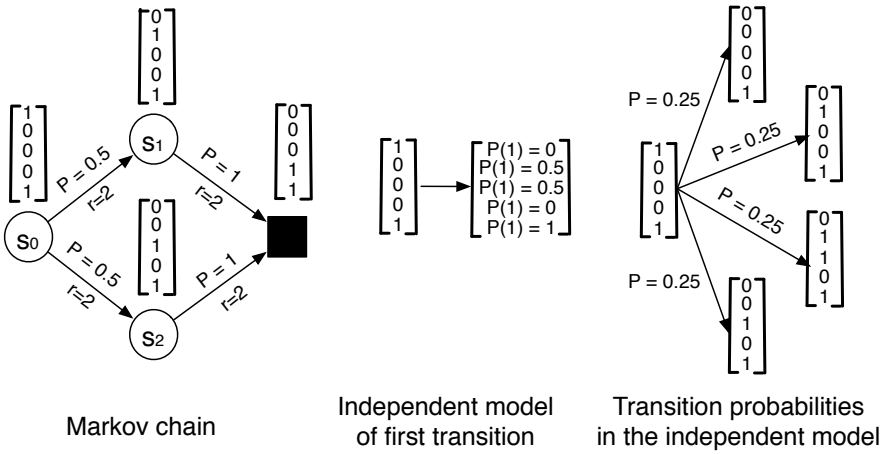

Figure 4.2: Independent sampling planning can be unsound for this small Markov chain.

4.9 Limitations of Independent Sampling

As previously mentioned, iplanner can be guaranteed to work only when the components of each feature vector are conditionally independent given the previous features and an action. The following example shows that, even with a perfect model, iplanner can diverge in a policy evaluation setting.

Take the Markov chain on the left side of Figure 4.2, where s_0 transitions to either s_1 or s_2 with probability 0.5. A correct model for independent sampling would state that, if the feature vector corresponding to s_0 is observed, the probabilities of individual components of the next feature vector being equal to 1 are $0, 0.5, 0.5, 0$ and 1 respectively. Sampling from this independent model would generate any of the four feature vectors to the right of Figure 4.2. From this, it can already be observed that the probabilities of these vectors being sampled do not correspond to the correct transition probabilities.

Even more disturbing is the fact that learning a value function from sequences of feature vectors generated this way can be unsound. There exist log-linear independent sampling models (not included here) that can generate independent probabilities consistent with the Markov chain in Figure 4.2. Among these models, some will cause $[0\ 0\ 0\ 0\ 1]^T$ to transition to itself with probability one. If such a model samples the initial transition $[1\ 0\ 0\ 0\ 1]^T \rightarrow [0\ 0\ 0\ 0\ 1]^T$, it will afterwards go into an infinite loop where $[0\ 0\ 0\ 0\ 1]^T$ is sampled forever. If the expected reward for transitioning from $[0\ 0\ 0\ 0\ 1]^T$ is non-zero, then the value function

estimated based on such a sequence will be unbounded.

In practice, sampled trajectories will normally be cut off after a finite number of steps, and thus divergence is unlikely to occur. Nevertheless, the phenomenon illustrated in the example above can negatively impact the performance of iplanner.

On the other hand, a similar problem does not arise for cascade sampling. A correct cascade sampling model will always sample from the correct distribution over feature vectors.

4.10 Conclusion

This chapter presented an outline of a generic system for sampling-based planning and learning with feature-based approximators, followed by the description of two particular instantiations of such a system.

The generic system was described in a structured, clear and general manner. This description emphasizes the flexibility that sampling-based planning allows in the choice of function approximators and learning algorithms.

The particular methods proposed define all components of the system, thus resulting in complete and working algorithms. Iplanner, the method based on independent sampling of each feature-vector component, is conceptually simple but has the disadvantage that it is limited in its representational power. On the other hand, cplanner, based on the so-called 'cascade sampling', is more general because it can represent dependencies between different components of the same feature vector.

In the following paragraphs, I will discuss how the cascade sampling architecture is related to existing methods for representing approximate generative MDP models.

Among current sampling-based MDP planning algorithms with approximate models, two main directions exist. The first represents the model using a particular, limited type of function approximation, such as state aggregation (Kuvayev and Sutton, 1996) or DBNs with conditional independence assumptions (Tadepalli & Ok, 1996; Andre, Friedman & Parr, 1998). The second direction can represent the model using arbitrary function approximation, at the cost of only describing the first two moments of the transition distribution: $E(\phi_{t+1}|\phi_t, a_t)$ is learned for each state using general function approximation, and a glob-

ally estimated uniform (Atkeson, Moore & Schaal, 1997) or normally distributed (Ng et al, 2004) noise is added to this expected value.

Compared to these, cascade sampling with binary features represents a third type of approach. General function approximation can be used for each component, and arbitrary probability distributions can be represented, as long as binary features that allow this generality can be constructed. Constructing appropriate binary features is not a trivial matter, but it is one that is often addressed anyway in the context of value function approximation. Another issue related to cascade sampling is that its performance might depend on the order in which components of the feature vector are sampled, and it is not yet clear how to design a good mechanism for ordering these components.

It should also be noted that cascade sampling with binary features is, in essence, a DBN representation, but its compactness stems from representing the probability of each component using function approximation rather than from conditional independence assumptions. Previous work exists on using DBNs with parametric function approximation for each component as MDP models in non-sampling planning systems (e.g. Sallans, 2002; Degris et al., 2006); the work in these papers still relies on conditional independence between the components of ϕ_{t+1}.

Finally, one should keep in mind that statistical density estimation is a huge field, and it is likely that other existing density estimation techniques can easily be adapted for representing approximate generative MDP models. For instance, the Boltzmann machine (e.g. Ackley, Hinton and Sejnowski, 1985) could be used to learn a joint generative model for binary state representations. While the original Boltzmann machine learning algorithm is considered to be slow and unreliable, more recent versions posing additional restrictions have been shown to be effective in practice (Welling and Hinton, 2002; Hinton, Osindero and Teh, 2006).

Chapter 5

Empirical Illustration

The empirical results presented in this chapter illustrate the behavior of the sampling-based planning systems described in Chapter 4. The first experiment shows that a planning method based on the cascade sampling architecture can be more data-efficient than model-free learning in a continuous, stochastic domain. The second experiment shows that this advantage is even greater if a sequence of tasks have to be solved in the same domain. Finally, the third experiment shows that arbitrary generalization can weaken the quality of planning methods relative to model-free methods.

5.1 Testbed for Experiments 1 and 2: The Soft Obstacle Domain

The soft obstacle domain is a simplistic simulation of a navigation task with continuous state and stochastic dynamics. The 2D environment, illustrated in Figure 5.1, includes two kinds of obstacles, soft and hard, whose effects will be explained shortly. The agent can be located at any position on the map except for the regions where hard obstacles are present. The (continuous) state is represented by the coordinates of the agent's current position, $x \in [0, 1]$ and $y \in [0, 1]$. There are four actions available at every time step: north, south, east and west. Each of the actions causes the agent to move for a distance of 0.05 on average in the corresponding direction, unless this movement would take the agent into an obstacle area or outside the map boundaries. A normally distributed noise of standard deviation 0.025 is added to the actions' effects in each direction. Actions that would cause the agent to move outside the map boundaries or inside a hard obstacle have no effect: the respective time step elapses without any movement. If the action causes the agent to move

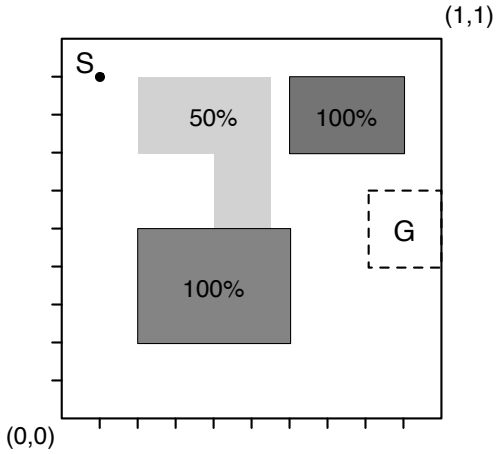

Figure 5.1: The soft obstacle domain. Attempts to enter the dark regions were rejected with 100% probability and for the light region with 50% probability.

to a position inside a soft obstacle, either the action has no effect (like in the case of hard obstacles), or the agent is moved to that position. Each of these two outcomes occurs with 50% probability.

The agent's objective is to navigate from the starting state to the goal region in a minimal number of time steps. This is formulated as an undiscounted, episodic problem, with the agent receiving a negative reward of -1 per time step until the goal is reached.

Tile coding was used to form the binary feature vectors ϕ_s for this domain. For all experiments 8 tilings were used, each roughly a 10-by-10 tiling of the 2D space. The features generated by the tile coding process were then hashed down to 400 in number using Rich Sutton's tile coding software[1]. Another 25 features were obtained by discretizing the state space into a 5-by-5 grid and taking the index of the grid cell corresponding to the current state; one of these grid locations corresponded to the goal region, and the agents had knowledge of what this 'terminal feature' was. The 25 extra features will be important for Experiment 2, and I have chosen to use the same feature sets over the two experiments. Finally, a feature that always took a value of 1 was added as a bias unit. The feature vectors were thus of length $n = 426$; exactly 10 components of each feature vector took a value of one, while the rest were equal to zero.

[1]Sutton's tile coding software is available at http://www.cs.ualberta.ca/∼sutton/tiles2.html

5.2 Experiment 1: Data-efficiency in a Single Task

As discussed in Section 2.4.2, previous empirical results for finite MDPs suggest that model-based methods can be more data-efficient than their model-free counterparts. The main experiment in this section investigates whether the same can be said for the sampling-based planning and learning methods proposed in Chapter 4. The experiment will use the soft obstacle domain, a continuous, stochastic problem.

5.2.1 Experiment Description

Three algorithms were applied to the soft obstacle domain: planning and learning with an independent sampling model (iplanner), planning and learning with a cascade sampling model (cplanner), and model-free reinforcement learning (equivalent to zero planning steps). All algorithms used linear value function approximation with the tile coding features described in Section 5.1.

Sarsa(0) was used to learn the parameters of the linear value function approximator. For the planning methods, Sarsa(0) was used to learn both from simulated data and from real data; for the model-free method, it was (obviously) only used with real data. The policy was ϵ-greedy with respect to the current value function, and $\epsilon = 0.1$ was used for all algorithms.

The length of each simulated trajectory was set to $L = 20$. If the feature corresponding to the terminal region was sampled at step k of this trajectory, then the rest $20 - k$ steps were not sampled. Only one trajectory was sampled at every time step ($N = 1$).

Several learning rates for the model and the value function were used in the experiments. For model-free Sarsa, the learning rates for the value function were $\alpha \in \{0.1, 0.2, 0.4, 0.8\}$, while for both planning methods the ranges were $\alpha \in \{0.1, 0.4\}$ and $\beta \in \{2, 4, 8, 16\}$. A reward learning rate of 0.1 was used, but this was of little importance as the rewards were -1 at any time step. These parameter values were divided by the number of non-zero features in the current feature vector.

Each algorithm was run for 100000 time steps and the whole process was repeated for 30 runs.

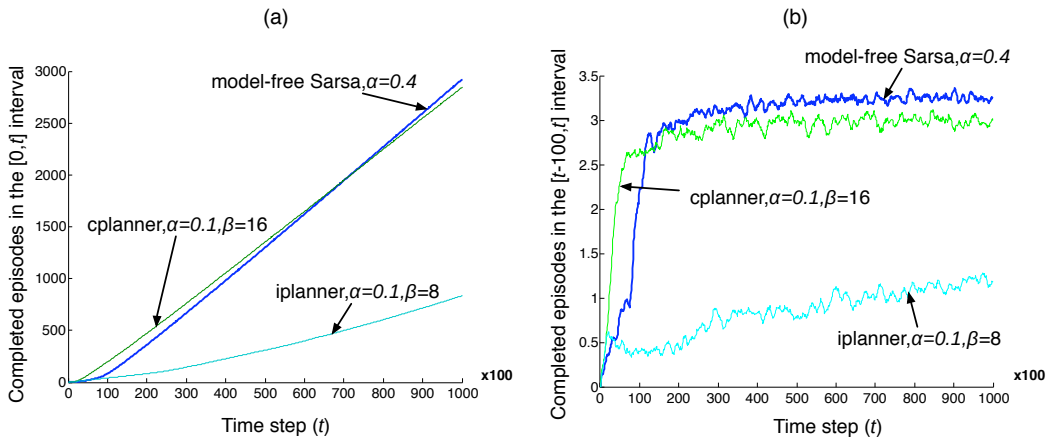

Figure 5.2: Total (a) and average (b) number of episodes completed in the single-task scenario. Results are averaged over 30 runs.

5.2.2 Results

The results for the best parameter settings for each algorithm are summarized by the graphs in Figure 5.2. The first graph shows, for each t, the average number of episodes completed during the $[0, t]$ interval. The second graph shows the average number of episodes completed during the $[t - 100, t]$ interval, and was smoothed with a window of size 10.

The iplanner algorithm clearly performed poorly. On the other hand, cplanner managed to make better use of initial experience than the model-free method: for all values of t between 200 and 10700, the policy learned by cplanner was better on average than the policy learned by the model-free method with the same amount of experience. The initial difference in performance is illustrated by the graphs in Figure 5.3.

Given enough experience, model-free learning found the best solution out of all the methods. During the last 10000 time steps, the average length of an episode was 30.8 steps for model-free learning (with a standard error of 0.01), 33.6 for cplanner (with a standard error of 0.02) and 84 for iplanner (with a standard error of 0.14).

5.2.3 Discussion

Using its best parameter settings, cplanner required less data to learn a good policy than the model-free algorithm. The intuitive explanation for this is that, by also learning the transition models, planning methods store more information about previous experience than

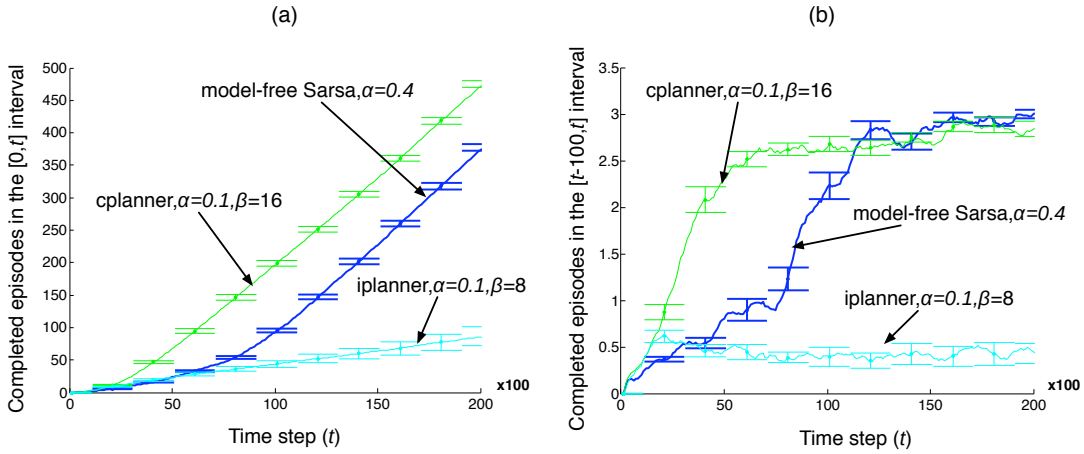

Figure 5.3: Total (a) and average (b) number of episodes completed in the single-task scenario over the first 20000 time steps. The error bars represent the standard error at selected points. Results are averaged over 30 runs.

a method that only uses a value function.

The poor performance of iplanner should not be surprising. Due to the specifics of the tile coding process, the features used in this experiment were highly correlated, while iplanner assumes conditional independence. In Section 4.9 it was demonstrated that iplanner can perform poorly even on a simple synthetic MDP if the features are correlated.

While planning can learn a good solution faster, the solution that model-free learning stabilizes to is better than the one that planning stabilizes to. Even after a lot of experience, the model learned by the planning methods is likely to be imperfect due to the function approximator. Thus, the agent will learn from trajectories that are sampled from a distribution that is not the correct distribution. The model-free method, on the other hand, uses only data from the real world, which is generated from the correct probability distribution. The ultimate performance of the model-free algorithms is limited only by the resolution of the function approximator for the value function, whereas the planning methods are also limited by the function approximator for the model.

The Effect of the Transition Model Learning-Rate

The learning rate for the transition model (β) had a significant effect on the results: cplanner showed better data-efficiency than model-free learning for $\beta = 16$, but not for smaller

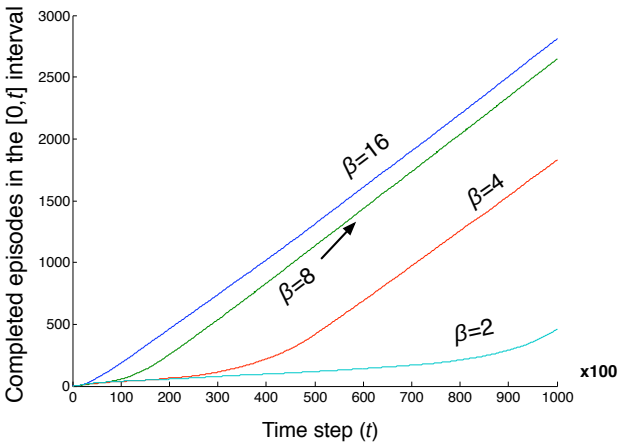

Figure 5.4: Total number of episodes completed by cplanner in the single-task scenario for different values of the model learning-rate (β). Results are averaged over 30 runs.

values of β. As it can be observed in Figure 5.4, cplanner requires more time to learn a good policy as the value of β decreases.

The role that the learning rate has in on-line gradient-descent methods, such as the one that cplanner and iplanner use, can explain this phenomenon. In on-line gradient-descent methods, larger values of the learning rate lead to bigger steps in the direction of the gradient, which can in turn lead to getting close to the optimal solution more quickly. The danger of using large learning-rates, however, is that the algorithm may not be able to stabilize to a solution that is sufficiently close to the optimum. The results seem to indicate that, for the soft obstacle domain, the model learned with a large learning-rate is sufficiently close to the correct model to allow for a good policy to be computed.

In general, however, a large learning-rate for the model can lead to clearly sub-optimal behavior. This is not evident from the soft obstacle domain experiment, and will be illustrated on the highly simplified domain in Figure 5.5 (called the 'soft pathway domain').

The state space for the soft pathway domain is extremely small: each state corresponds to one of the grid locations illustrated in Figure 5.5. The four available actions (north, south, east and west) move the agent to the corresponding adjacent grid cell, unless one of the following situations occurs: if the action would take the agent outside the boundaries or into a hard obstacle, that action has no effect; if the action would take the agent into a state inside the soft obstacle, then either the action has no effect or the agent moves to that

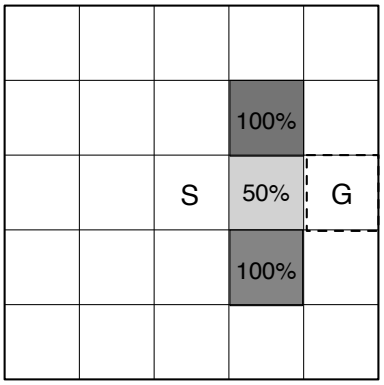

Figure 5.5: The soft pathway domain, used to illustrates that aggressive online gradient descent (large value of the β parameter) can lead to sub-optimal behavior even in a very simple task. Each grid cell in the figure represents a state.

state, each with probability 50%. The name 'soft pathway domain' indicates that there is a direct pathway from start to goal going between the two hard obstacles and through the soft obstacle.

Consider now running a planning and learning algorithm such as cplanner on the soft pathway domain. Because of the size of the state space, the model does not need to be approximated, and can be stored for instance in a table. Thus, the learned model will converge to the correct model if β is appropriately decreased. If β is too large, however, then convergence cannot be guaranteed and planning with the learned model can lead to sub-optimal behavior.

In the rather extreme case of $\beta = 16$, for instance, the agent learns an almost-deterministic transition model, practically always giving full credit to the last observed transition. This means that the agent's learned model will predict that the outcome of taking action a at state s will always be the last observed outcome of taking a at s.[2] This is reminiscent of memory-based methods such as Lin's experience replay (Lin, 1993). In the soft pathway domain, if the last attempt to go east from the starting state had no effect, then all the sampled trajectories from the learned model would predict that going east from the starting state has no effect. Thus, the policy that the agent learns from these trajectories will not consider going through the soft obstacle to be a viable alternative, and will instead search

[2]This is not necessarily true if function approximation is used; because of generalization, the prediction for a certain state may change even after the last visit to that state.

for an alternate route.

This was verified empirically by running cplanner with $\beta = 16$ and $\beta = 1$ on the soft pathway domain (the other parameters were $\alpha = 0.1$, $\beta^R = 0.1$, $N = 1$ and $L = 20$). The two variants of cplanner were run for 20000 time steps. Over the last 10000 time steps, the average length of an episode under the policy computed by cplanner with $\beta = 16$ was 4.47, while for $\beta = 1$ it was 2.49 (results were averaged over 30 runs). This discrepancy occured, as expected, because cplanner with $\beta = 16$ often avoided the shorter path through the soft obstacle.

To conclude this section, the soft obstacle domain results emphasize the fact that fast, data-efficient model-learning can help the data-efficiency of a planning and learning system. In the soft obstacle domain, fast model-learning was achieved by simply using a large value of β. The soft pathway domain results, however, illustrate that this is not always an adequate solution. Investigating a more reliable solution for data-efficient learning of the generative model is a subject of future work.

5.3 Experiment 2: Data-efficiency over a Sequence of Tasks

The second experiment illustrates the advantages of model-based methods in the context of different tasks in the same environment. A simple formulation of this multi-task scenario has been used, where the terminal state is changed at some point in time and the agent has knowledge of the change. More general formulations, such as modifying the reward function without notifying the agent of the change, are not considered here.

5.3.1 Experiment Description

The soft obstacle domain was also used for this experiment, this time in a two-task scenario. The agent had to solve a sequence of two tasks, each of them with a different goal region. As illustrated in Figure 5.6, the goal region for the first task is G_1, while the goal region for the second task is G_2.

The cplanner, iplanner and model-free Sarsa(0) algorithms were ran for 100000 time steps each with the same parameters as for Experiment 1, except that only the two best values of β (8 and 16) and the best value of α (0.1) were used for the planning methods.

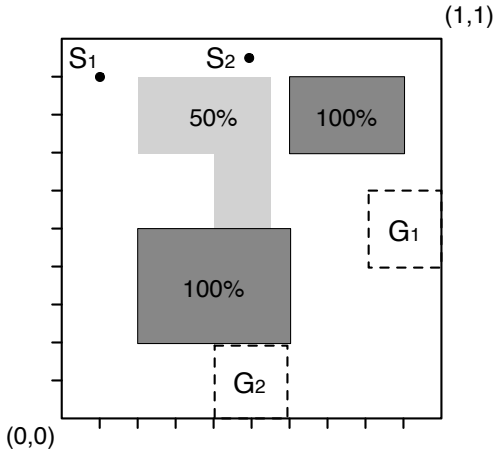

Figure 5.6: The two-task scenario for the soft obstacle domain. Halfway through the learning process, the starting state was changed from S_1 to S_2 and the goal state was changed from G_1 to G_2.

Halfway through the time interval (after 50000 time steps) the goal region was changed from G_1 to G_2 (i.e. the episodes terminated when the agent reached G_2 rather than G_1). At the same time, the starting position changed from S_1 to S_2. The agent had knowledge of this change via the special terminal feature: at time-step 50000, the agent was provided the information that the terminal feature is the one corresponding to locations in G_2 rather than G_1. Note that a new feature corresponding to G_2 did not need to be created, because it already existed because of the 5-by-5 tiling described in Section 5.2.1

When the goal region changed, all agents re-set their value functions to zero. The planning agents kept the already learned transition and reward model, and performed additional planning by generating 1000 episodes from the learned model before resuming normal interaction with the environment. Planning was obviously not a possibility for the model-free agent, so that agent resumed interaction immediately.

5.3.2 Results

For each algorithm, I measured the average ratio of episodes completed during the first 10000 steps of the second task to episodes completed during the first 10000 steps of the first task. This ratio was 1.45 for cplanner with $\beta = 16$, 2.39 for cplanner with $\beta = 8$, 1.51 for iplanner and 1.005 for model-free Sarsa(0) with $\alpha = 0.4$ (the best parameter setting

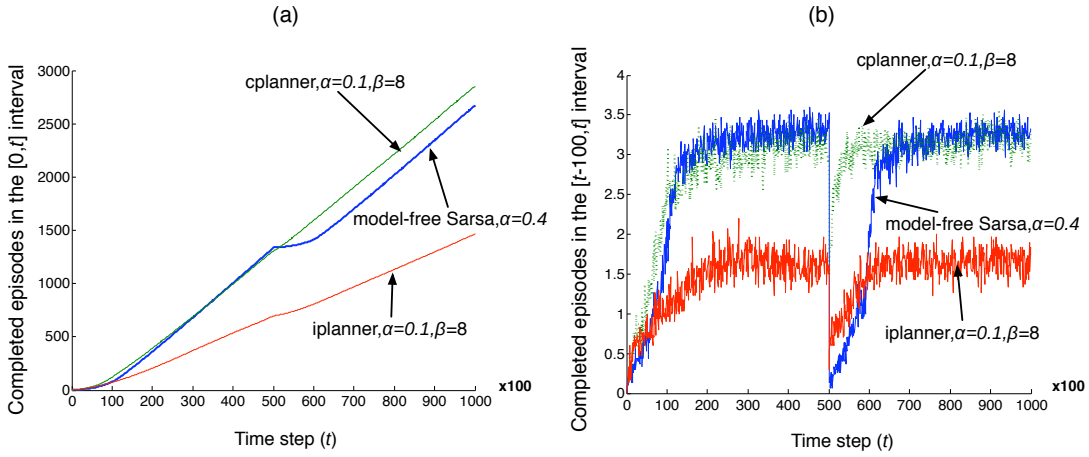

Figure 5.7: Total (a) and average (b) number of episodes completed in the two-task scenario. Results are averaged over 30 runs.

in terms of total number of episodes completed over the two tasks). This indicates that, while the two tasks were of similar difficulty (the performance of model-free learning was virtually identical), the planning methods were more adept at solving the second task after having gained experience on the first.

Similarly to Experiment 1, the graphs in Figure 5.7 show the average of the number of episodes completed in the $[0, t]$ interval (Figure 5.7(a)) and in the $[t-100, t]$ interval (Figure 5.7(b)). Among the two parameter settings used for cplanner, only the learning curve for $\beta = 8$ was included, as the ratio measured in the paragraph above was better for $\beta = 8$ than for $\beta = 16$. The results show that, after the goal region is changed, cplanner's performance drops less and recovers more quickly than model-free learning.

5.3.3 Discussion

The results illustrate that model-based methods are particularly appropriate for multi-task scenarios, since experience accumulated while solving the first task(s) allow them to perform better on subsequent tasks. Starting with the second task, model-based methods can use the transition model learned while solving previous tasks; in contrast, model-free methods only learn a value function or a policy, and both of these quantities are task-dependent and therefore non-reusable.

5.4 Experiment 3: The Effect of Arbitrary Generalization

All compact representation methods induce some form of generalization. For instance, the model representation architectures introduced in Chapter 4 generalize between states with similar feature vectors: the more non-zero features two vectors ϕ_1 and ϕ_2 have in common, the more of the weights used in computing the approximate values for $P(\cdot, a, \phi_1)$ and $P(\cdot, a, \phi_2)$ will be the same, thus causing the approximate transition probabilities to have similar values.

The previous comment suggests that the success of the planning methods will depend on the extent to which the transition probabilities corresponding to similar feature vectors are also similar. An empirical approach to supporting this idea is taken here.

The following experiment illustrates how generalization induced by feature vectors that are arbitrary and have no relationship to the problem structure can be harmful to planning methods. The results indicate that arbitrary generalization hurts planning methods more than it hurts model-free methods; this can be explained by the fact that for planning methods generalization is involved in learning both the model and the value function, while for model-free methods it is only involved in learning the value function.

5.4.1 Experiment Description

A suite of randomly generated MDPs was used to investigate the effects of arbitrary generalization. Each MDP had 100 states and 5 available actions at each state. For each state s and action a, two successor states s'_1 and s'_2 were randomly selected out of the remaining 99 states without repetition, and the transition probabilities $P(s'_1, a, s)$ and $P(s'_2, a, s)$ were assigned random values between zero and one such that $P(s'_1, a, s) + P(s'_2, a, s) = 1$. The expected reward $R(s, a)$ was randomly generated from a normal distribution $\mathcal{N}(0, 1)$. The reward that the agent observed after taking action a in state s was the expected reward $R(s, a)$ plus a normally distributed noise of variance 0.1. The objective was to find a policy that maximizes the average of future rewards.

The states were observed by the agent via binary feature vectors. For all experiments, these feature vectors were 100-dimensional. The number of non-zero features was varied in order to illustrate the effect of generalization: for each state $s \in \{0, ..99\}$ the feature vector

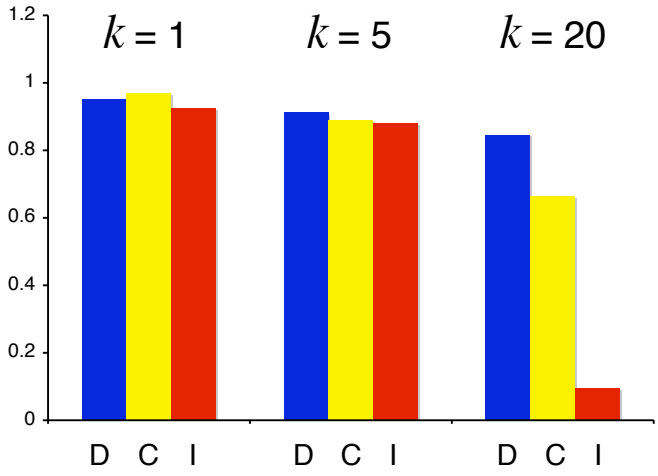

Figure 5.8: Average reward per time step obtained by the three algorithms (D=direct, C=cplanner, I=iplanner) over 100000 time steps on a sequence of 30 random MDPs, for different values of the number of non-zero features k.

ϕ_s had the k components s, $(s+1) \mod |S|$, ... $(s+i) \mod |S|$ equal to one and the rest of the components equal to zero. The parameter k was given the values $1, 5$ and 20.

Similarly to the previous experiments, three algorithms were used: cplanner, iplanner and a direct (model-free) learning algorithm. The only difference was that, since this was an average-reward problem, all algorithms used R-learning (Section 2.3.1) for the policy-update step[3]. The step-size α_ρ used for estimating the average reward was set to 0.001 for all algorithms. For cplanner and iplanner, all combinations of $\alpha \in \{0.05, 0.1, 0.2\}$ and $\beta \in \{1, 4, 16\}$ were tried, while the reward model learning-rate was arbitrarily set to $\beta^R = 0.1$. For model-free learning, $\alpha \in \{0.05, 0.1, 0.2\}$ was used. The results displayed are for the best parameter values for each algorithm. Similarly to Experiments 1 and 2, the number of sampled trajectories was $N = 1$ and the length of each trajectory was $L = 20$.

5.4.2 Results

Figure 5.8 shows the average reward per time step obtained by the three algorithms during 100000 time steps, averaged over 30 runs. Each run used a different randomly generated MDP, but the same sequence of MDPs was used for all algorithms.

It can be observed that direct RL gained more of an advantage relative to the planning methods as the number of non-zero features increased. This advantage was especially evident when 20-dimensional feature vectors were used.

5.4.3 Discussion

The results indicate that arbitrary generalization hurts the performance of all methods. However, they also indicate that the performance of the planning methods is hurt more than the performance of model-free learning. My explanation for this is that model-free learning was only affected by arbitrary generalization in the value-function, while the planning methods were affected by arbitrary generalization for both the model and the value function.

It should be emphasized that the value of k only influenced the amount of generalization and not the discrimination power. Normally, poor discrimination power due to the fact that function approximation is used for the model can be another factor that causes planning methods to underperform model-free methods. In the experiments presented here, however, the features were constructed such that the vectors $\{\phi_1, \phi_2, ..., \phi_{100}\}$ were linearly independent. This, together with the fact that the logistic function is invertible, means that both the model and the value function could be represented exactly for all values of k. Thus, superior discrimination power was not what caused model-free learning to gain the advantage as the number of non-zero features increased.

Generalization, on the other hand, did indeed occur when $k > 1$, and it occurred in an arbitrary fashion. As explained earlier, the model built by cplanner and iplanner generalizes between feature vectors with non-zero components on the same positions. The bigger the value of k, the more non-zero components different states had in common, and thus

[3]For simplicity, I have abused notation and used the names cplanner and iplanner even though R-learning is used instead of Sarsa

the more generalization occurred. This generalization was arbitrary: the MDP model was randomly generated, and the feature vectors were simply assigned based on the state labels, completely ignoring the dynamics of the MDP. In contrast, a potentially meaningful feature assignment scheme would associate similar feature vectors to states with similar transition probabilities.

This section offers empirical support for the idea that the success of planning algorithms such as cplanner depends on the availability of feature vectors that are related to the MDP's dynamics. In the extreme case presented here, generalization based on the random feature assignment scheme caused cplanner and iplanner to perform worse than model-free learning.

Finally, this experiment is offering further evidence of the flexibility of the sampling-based planning framework. The exact same function approximator and learning algorithm as in the experiments on the soft obstacle domain were used for the model, but a different algorithm (R-learning) was used for learning the value function. Yet, the R-learning algorithm was smoothly integrated with the model learning and approximation methods.

5.5 Limitations

The results presented in this chapter have not been compared with other methods for planning and learning with approximate models (e.g. Sallans, 2002; Degris et al., 2006; Kuvayev and Sutton, 1996). Although these methods use more restrictive model approximators than cplanner, it is still possible that they are successful on a wide range of problems, for instance on the soft obstacle domain. Therefore, the lack of such a comparison is a limitation of this chapter.

The experiments presented here do not include extensive parameter optimization. The main reason for this is the large number of parameters to be optimized – up to seven for cplanner or iplanner with R-learning. Besides being a limitation of the current experiments, this highlights the fact that optimizing the parameters of the planning methods for particular problems may require a considerable effort.

5.6 Conclusion

Experiments 1 and 2 have shown that model-based methods can be more data-efficient than model-free methods even in continuous and stochastic domains, particularly if the agent has to solve a sequence of tasks in the same environment. When deciding what algorithm to use in practice, data-efficiency is an important criterion, especially for domains such as robotics where gathering experience is hard and costly.

Nevertheless, other criteria besides data-efficiency might be important for selecting what method to use. Experiment 3 illustrated the negative effect that arbitrary generalization can have on planning methods Another important aspect is computational complexity. In general, model-based methods will require more resources than their model-free counterparts – recall from Chapter 4 that the time complexity of cplanner and iplanner is $O(NLnk)$ for every time step, whereas Sarsa(0) only requires $O(k)$ operations per time step. Because of this, model-free methods can work with a much larger (and therefore potentially more expressive) feature space than model-based methods if little computation time is available per time step. Thus, a judicious decision would involve considering factors such as computational resources, the difficulty of collecting data or the expressiveness of the feature set.

The results also showed that the model-free method found a better final solution than the model-based methods. This can be explained by the fact that model-based methods eventually pay tribute to inaccuracies in the model caused by function approximation. Still, this fact is slightly disturbing, since all methods used the same `policy-learn` algorithm and the same experience; a possible solution to this problem could be to give less credit to the model as more experience is accumulated.

Chapter 6

On the Possibility of Expectation-based Planning

This chapter investigates the possibility of planning when the model only contains the expected value of the next state (feature vector) instead of a complete distribution over next states. Expectation-based planning is appealing because, as explained in Section 6.4 , representing and learning expectation-based models is considerably easier than representing and learning the complete transition distribution. The results in the following sections offer new insights about the potential and limitations of expectation-based planning.

Note that the results in this chapter hold for discounted cumulative return problems. Some of them appear to be easily extendible to the average-reward case, but this extension will be left for future work.

6.1 Value Iteration with Expectation-based Models

The results in this section use a feature-based, approximate version of expectation-based, action-conditioned, one-step MDP transition models. An expectation-based, action-conditioned, one-step MDP transition model predicts $E[s_{t+1}|s_t = s, a_t = a]$, the expected value of the next state s_{t+1} given current state s_t and action a_t. A feature-based, approximate version of such a model predicts the expected value of the next feature vector $\phi_{t+1}(i)$ given ϕ_t and a_t:

$$E[\phi_{t+1}|\phi_t = \phi, a_t = a] = f(\phi, a),$$

where f is the feature-based function approximator.

As described in Section 2.4, computing the Bellman operator is the main step of the value iteration class of MDP planning methods [1]. The key observation for this section is that, for a certain class of value function approximators, the Bellman operator can be computed using an expectation-based model of the MDP. The class of function approximators is described by the following assumption:

Assumption 1. *Given the feature-generating mechanism $\phi : S \to \Phi$, the value function approximator $f : \Phi \to \mathbb{R}$ is such that, for any $s \in S$ and $a \in A$,*

$$\int V(s')P(ds', a, s) = f(E[\phi_{s'}|s_t = a, a_t = a])$$

Observation 1. *Assumption 1 is verified if the value function approximator f is linear in the features. To see why, simply observe that if $f(\phi) = v^T\phi$ then*

$$\int V(s')P(ds', a, s) = \int v^T\phi_{s'}P(ds', a, s) = v^T E[\phi_{s'}|s, a]$$

If Assumption 1 is verified, it immediately follows that the Bellman operator can be computed using an expectation-based model. Under Assumption 1, the Bellman operator can be re-written as

$$BV(s) = \max_{a \in A} [R(s, a) + \gamma f(E[\phi_{s'}|s, a]])$$

or, for the particular case of linear value function approximation,

$$BV(s) = \max_{a \in A} \left[R(s, a) + \gamma v^T E[\phi_{s'}|s, a]\right]$$

This observation opens the door to approximate value iteration algorithms where not only the value function, but also the model is approximate. Carefully designed adaptations of existing algorithms (e.g. de Farias and Van Roy, 2000; Munos and Szepesvari, 2005) could form a new class of sound, yet practically efficient methods for MDP planning with learned and approximate models.

[1] Recall that the Bellman operator B was defined by

$$BV(s) = \max_{a \in A} \left[R(s, a) + \gamma \int V(s')P(ds', a, s)\right], \forall s \in S.$$

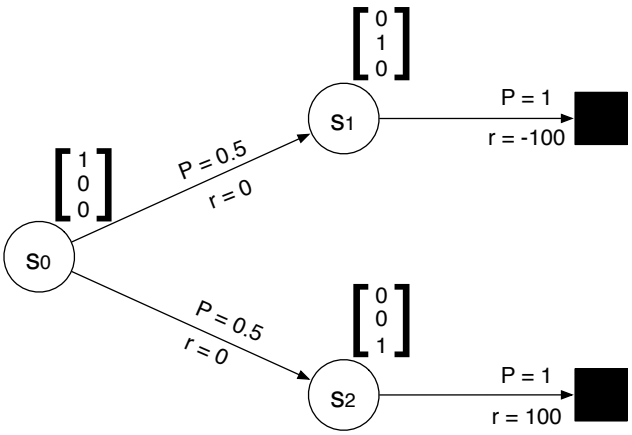

Figure 6.1: MDP used to illustrate a limitation of expectation-based models.

6.2 Learning Multiple State-Values from a Single Projection

In sampling-based planning, a trajectory sampled starting from a particular state allows the agent to update not only the value of that state, but also the values of all the other states along the trajectory. Sampling-based methods can thus estimate the value function of all states in S while requiring a single state to be provided as input. This property can be extremely useful: for instance, if a model is available then the value of any new policy can be computed without having to interact with the environment using that policy.

In this section I will show that expectation-based models do not have a similar property. Assume a perfect expectation-based model, one that would compute, for any policy π and initial state s, the correct values of $E[r_i|s_0 = s, \pi]$ and $E[s_i|s_0 = s, \pi]$ for all $i \geq 0$. These may still not contain sufficient information for computing the value of any state other than s_0.

To illustrate this, consider the simple MDP in Figure 6.1. This MDP has three non-terminal states s_0, s_1 and s_2, and only one possible action a. The state representation consists of a unique unit-basis vector for each state, as indicated in the figure (no features are required for the terminal states, since their value will always be zero). Starting in s_0, the agent will always observe the sequence of expected state vectors $[1\ 0\ 0]^T, [0\ 0.5\ 0.5]^T$. The sequence of expected rewards is $0, 0$.

These sequences contain sufficient information for correctly computing $V(s_0) = 0$.

They do not, however, contain enough information for computing $V(s_1)$ and $V(s_2)$: because the projection is performed in an average, expected fashion, the elements in the projection do not reflect the fact that the agent receives a reward of -100 after $[0\ 1\ 0]^T$ is observed, and a reward of 100 after $[0\ 0\ 1]^T$ is observed.

6.3 Multi-step Projection with Expectation-based Models

In this section I will show that a linear expectation-based model of policy π can be used to compute $V^\pi(s)$ at any given state s. An expectation-based, policy-conditioned model for policy π predicts

$$E[s_{t+1}|s_t = s, \pi] = f_\pi(s).$$

The following results require that the policy-conditioned expectation-based model is linear:

$$f_\pi(s) = M_\pi s,$$

where M_π is a n-by-n matrix.

Lemma 1. *If*

$$E[s_{t+1}|s_t = s, \pi] = M_\pi s, \forall t \geq 0, \forall s \in S$$

then for any $n \geq 0$

$$E[s_{t+n}|s_t = s, \pi] = M_\pi^n s, t \geq 0, \forall s \in S$$

Proof. (The following holds for $n = 2$; the proof for the general case is similar.)

$$\begin{aligned} E[s_{t+2}|s_t = s, \pi] &= E_{s_{t+1}}\left[E[s_{t+2}|s_{t+1}, s_t = s, \pi]|s_t = s, \pi\right] \\ &= E_{s_{t+1}}\left[E[s_{t+2}|s_{t+1}, \pi]|s_t = s, \pi\right] \text{ (Markov property)} \\ &= E_{s_{t+1}}\left[M_\pi s_{t+1}|s_t = s, \pi\right] \\ &= ME[s_{t+1}|s_t = s, \pi] = M_\pi^2 s \end{aligned}$$

□

Lemma 2. *If*

$$E[s_{t+1}|s_t = s, \pi] = M_\pi s, t \geq 0, \forall s \in S$$

and

$$E[r_{t+1}|s_t = s, \pi] = U_\pi s, t \geq 0, \forall s \in S$$

then for any $n \geq 0$

$$E[r_{t+n}|s_t = s, \pi] = U_\pi M_\pi^{n-1} s, t \geq 0, \forall s \in S$$

Proof.

$$\begin{aligned} E[r_{t+n}|s_t = s, \pi] &= E_{s_{t+n-1}}\left[E[r_{t+n}|s_{t+n-1}, s_t = s, \pi]|s_t = s, \pi\right] \\ &= E_{s_{t+n-1}}\left[E[r_{t+n}|s_{t+n-1}, \pi]|s_t = s, \pi\right] \text{ (Markov property)} \\ &= E_{s_{t+n-1}}\left[U_\pi s_{t+n-1}|s_t = s, \pi\right] \\ &= U_\pi E[s_{t+n-1}|s_t = s, \pi] = U_\pi M_\pi^{n-1} s \end{aligned}$$

(the last equality is true because of Lemma 1) □

From the previous lemmas, it immediately follows that:

Theorem 3. *Linear one-step expectation-based transition and reward models for policy* π *are sufficient for computing* $V^\pi(s_s)$ *for any state* s

Proof. Using Lemma 1 and Lemma 2,

$$\begin{aligned} V^\pi(s) &= E\left[\sum_{i=0}^{\infty} \gamma^i r_{i+1}|s_0 = s, \pi\right] \\ &= \sum_{i=0}^{\infty} E\left[\gamma^i r_{i+1}|s_0 = s, \pi\right] = \sum_{i=0}^{\infty} \gamma^i U_\pi M_\pi^i s \end{aligned}$$

□

Theorem 3 shows the potential of policy-dependent expectation-based models. The obvious problem with policy-dependent models is that a new model has to be constructed every time the policy changes. Nevertheless, a practical implementation based on Theorem 3 could still be data-efficienct for evaluating a single policy. In this case, it should be compared with existing data-efficient methods for policy evaluation, such as LSTD. If an algorithm for computing the policy-conditioned model of any policy from action-conditioned models existed, then Theorem 3 could also be used in a policy improvement setting. Investigating under what conditions such an algorithm can be designed is an interesting topic for future work.

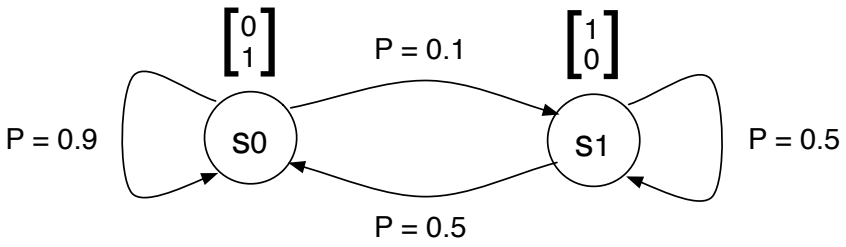

Figure 6.2: MDP used to illustrate potential problems with using non-linear expectation-based models.

6.3.1 Non-linear Models

The above proofs only hold for linear models, and, unfortunately, the results cannot be easily generalized to handle arbitrary function approximation for the model. I will present a simple example of an MDP for which an expectation-based non-linear model of the form $E[s_{k+1}|s_k = s, \pi] = f_\pi(s)$ cannot be used for multi-step projections: $E[s_{k+2}|s_k = s, \pi] \neq f_\pi(f_\pi(s))$.

Consider the MDP in Figure 6.2, with $S = \{[0\ 1]^T, [1\ 0]^T\}$ and $A = \{a\}$ (rewards are not important in this example). For this MDP, $E[s_{t+1}|s_t = [0\ 1]^T] = [0.1\ 0.9]^T$ and $E[s_{t+1}|s_t = [1\ 0]^T] = [0.5\ 0.5]^T$. A possible form for the function f_π (or simply f, since there is only one possible policy) is log-linear in the features, $f(s) = \sigma(Ms)$ where M is a 2-by-2 matrix of parameters. For our MDP, this choice of f can perfectly model $E[s_{t+1}|s_t = s]$ for $s \in S$ if we set

$$M = \begin{pmatrix} 0 & -2.1972 \\ 0 & 2.1972 \end{pmatrix}$$

However, this model cannot be directly used for multistep predictions. For instance, $E[s_{t+2}|s_t = s_{s1}] = [0.3\ 0.7]^T$, while $f(f(s_{s1})) = \sigma(M\sigma(Ms_{s1})) = [0.25\ 0.75]^T$. A linear model, on the other hand, would be able to make all multi-step predictions accurately.

6.4 Approximating and Learning Expectation-based Models

Learning an approximate one-step, action-conditioned, expectation-based model is an instance of the well-studied regression problem (e.g. Mitchell, 1997; Hastie, Tibshirani and Friedman, 2001): the agent has to learn a compact representation of $E[s_{t+1}|s_t = s, a_t = a]$,

having access to training data of the form $(s_t, a_t) \rightarrow s_{t+1}$. The problem of learning an expected reward model is similar. In the case of full probability models, all learning methods must make assumptions about the form of the probability distribution.

Note that the expected value of a vector is equal to the vector of expectations: for any vector ϕ, $E\left[[\phi(1)...\phi(n)]^T\right] = [E[\phi(1)]...E[\phi(n)]]^T$. Thus, unlike in the case of full probabilistic models, there is no need to worry about dependencies between different components at the same time step; predicting the expected value of each feature independently is sufficient for predicting the next vector.

6.5 Conclusion

This chapter analyzed, through a series of proofs and counterexamples, the opportunity of planning with expectation-based models.

A common theme throughout the chapter was the interplay of linearity and expectations. It was shown that if the value function is linear, then expectation-based models can be used as part of value iteration algorithms, and that a linear policy-dependent model is sufficient (at least theoretically) for computing the value function of any state.

An important limitation of expectation-based planning is that, unlike for sampling-based planning, the information provided by an expectation-based model is not sufficient for completely computing a value function; it is only sufficient for updating the value of a given state. This was illustrated by the counterexample in Section 6.2. Thus, for practical applications of expectation-based methods, states have to be generated using a separate process, perhaps by interacting with the environment or from a generative model.

Chapter 7

Conclusion

This thesis addresses the challenging problem of MDP planning with approximate, stochastic and learned models. Its primary contribution is proposing and empirically evaluating a sampling-based planning and learning system that is more general than existing methods for planning with approximate models. In addition, the theoretical results in Chapter 3 offer new insights into the possibility of planning with approximate expectation-based models, which are easier to learn and represent than general sampling-based models. The soundness of MDP planning with approximate models is theoretically demonstrated in Chapter 3.

Sampling-based planning is an appealing planning strategy, offering great flexibility in terms of choosing the function approximator for the model and the policy. The only constraint relating the two approximators is that they must use the same state representation.

Two ideas are key to the generality of cplanner, the sampling-based planning and learning system described in Section 4.7. First, transforming the state representation into binary feature vectors allows arbitrary function approximation to be used for representing the distribution of each component. Second, the model that cplanner learns allows for arbitrary dependencies between components of the feature vectors to be represented. Iplanner, a sampling-based planning system that ignores some of these dependencies, has been shown to be unsound.

The results in Section 5.3 illustrate the important point that using planning methods is particularly advantageous when a sequence of tasks has to be solved in the same environment. The fact that the models used were independent of the policy and the reward structure was the key to obtaining this advantage.

The results in Chapter 6 show that there are reasons to be optimistic about planning with approximate expectation-based models. Essential to the positive results about expectation-based planning was the fact that linear operators commute with the the expectation operator.

7.1 Limitations (Future Work)

The current empirical results are rather limited, and there are many ways to make them more illustrative and thorough. Analyzing the effect of individual parameters and using more domains, such as the ones available in the RL Library (White, 2006) are obvious extensions. An interesting scenario to experiment with would change the rewards or the transition model without the agent's knowledge, and test whether a model-based algorithm adapts to the changes faster than a model-free algorithm.

The algorithms used for model-learning (gradient descent) or policy-learning (Sarsa(0)) as part of cplanner are rather rudimentary. Because data-efficiency is the main reason for using planning methods, it is particularly important that the model-learning algorithms are data-efficient. By analyzing the literature on building and learning statistical models, it is likely that improvements on the gradient descent algorithm used by cplanner and iplanner can be made.

Chapter 6 only contains preliminary ideas and results about expectation-based planning. The obvious next step there is to transform these ideas into complete algorithms, and to analyze these algorithms empirically and theoretically.

Bibliography

Ackley, D. H., Hinton, G. E., Sejnowski, T. J. (1985). A Learning Algorithm for Boltzmann Machines. *Cognitive Science, 9:147-169*.

Andre, D., Friedman, N., Parr, R. (1998). Generalized Prioritized Sweeping. In *Advances in Neural Information Processing Systems 10*.

Antos, A., Szepesvari, C., Munos, R. (2006). Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. In *Proceedings of The Nineteenth Annual Conference on Learning Theory*.

Atkeson, C. (1993). Using local trajectory optimizers to speed up global optimization in dynamic programming. *Advances in Neural Information Processing Systems, 5*, 663–670. San Mateo, CA: Morgan Kaufmann.

Atkeson, C., Moore, A.W., Schaal, S. (1997). Locally Weighted Learning for Control. *AI Review*, 11: 75-113, Kluwer Acadedmic Publishers

Bengio, Y., Bengio, S. (2000). Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems 12*, pages 400-406.

Bertsekas, D. and Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Boutilier, C., Dearden, R., Goldszmidt, M. (2000). Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence 121*: 49107.

Boyan, J. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, 49:233–246.

Bradtke, S., Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57.

Cassandra, A. (1994). *Optimal Policies for Partially Observable Markov Decision Processes*. Technical Report CS-94-14, Brown University, Department of Computer Science.

Dean, T., Wellman, M.P. (1991) *Planning and Control*. Morgan Kaufmann Publishers.

Degris, T., Sigaud, O., Wuillemin, P. (2006). Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. *Proceedings of the 23rd International Conference on Machine Learning.*

de Farias, D.P. and Van Roy, B. (2000). On the Existence of Fixed Points for Approximate Value Iteration and Temporal-Difference Learning. *Journal of Optimization Theory and Applications*, Vol. 105, No. 3

Devroye, L. and Györfi, L. (1985). *Nonparametric Density Estimation: the L1 View*. Wiley, New York.

Geramifard, A., Bowling, M., Sutton, R. S. (2006). Incremental Least-Squares Temporal Difference Learning. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (AAAI), pages 356-361.

Ghallab M., Nau D., Traverso P. (2004) *Automated Planning, Theory and Practice*. Elsevier, Morgan Kaufmann Publishers

Guestrin, C., Patrascu, R., Schuurmans, D. (2002). Algorithm-Directed Exploration for Model-Based Reinforcement Learning in Factored MDPs. *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 235–242.

Guestrin, C., Koller, D., Parr, R. and Venkataraman, S. (2003). Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research (JAIR)19:399-468*.

Guestrin, C., Hauskrecht, M. and Kveton, B. (2004). Solving Factored MDPs with Continuous and Discrete Variables. In *Twentieth Conference on Uncertainty in Artificial Intelligence* (UAI 2004).

Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

Hinton, G. E., Osindero, S. and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18:1527-1554* .

Kaelbling, L.P. (1993). *Learning in Embedded Systems*. MIT Press.

Kaboli, A., Bowling, M. and Musilek, P. (2006). Bayesian calibration for Monte Carlo localization. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (AAAI) , pages 964–969.

Kakade, S., Kearns, M. and Langford, J. (2003) Exploration in Metric State Spaces. *Proceedings of the 20th International Conference on Machine Learning*

Kalmar, Z., Szepesvari, C. and Lorincz, A. (1998). Module-Based Reinforcement Learning: Experiments with a Real Robot. *Machine Learning 31:55-85*.

Kearns, M., Mansour, Y, and Ng, A.Y. (1999). A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1324–1331.

Kearns, M. and Singh, S. (2002). Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49:209-232.

Kuvayev, L., Sutton, R.S. (1996). Model-based reinforcement learning with an approximate, learned model. *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, pp. 101-105, Yale University, New Haven, CT.

Lagoudakis, M.G. and Parr, R. (2003). Least-Squares Policy Iteration. *Journal of Machine Learning Research* 4:1107-1149.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8:293-321.

Littman, M. (1996). *Algorithms for Sequential Decision Making*. Department of Computer Science, Brown University.

Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E. (2004) Inverted autonomous helicopter flight via reinforcement learning *International Symposium on Experimental Robotics*

Nouri, A., Littman, M. L. (2006). Investigating Function Approximation for Model-based Reinforcement Learning. Unpublished note at http://www.cs.rutgers.edu/ mlittman/ftp/ nouri-model-fa.ps.

Peng, J., Williams, R.J. (1993) Efficient learning and planning within the Dyna framework, *Adaptive Behavior 1*, 437–454.

Sallans, B. (2002). *Reinforcement Learning for Factored Markov Decision Processes*. Ph.D. Thesis, Dept. of Computer Science, University of Toronto.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 298-305.

Singh, S.P. (1992). Reinforcement learning with a hierarchy of abstract models. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 202–207. MIT Press.

Stone, P., Sutton, R. S., Kuhlmann, G. (2005). Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behavior*.

Sturtevant, N. and White, A. (2006). Feature Construction for Reinforcement Learning in Hearts. In *Computers and Games*.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224.

Sutton, R.S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding title. In *Advances in Neural Information Processing Systems 8*, pages 1038-1044.

Sutton, R.S., Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Szepesvari, C. and Munos, R. (2005). Finite Time Bounds for Sampling Based Fitted Value Iteration *Proceedings of the 22nd International Conference on Machine Learning*, pp. 881–886.

Tesauro, G.J. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38:58-68.

Tadepalli, P. and Ok, D. (1996). Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *Proceedings of the 13th International Conference on Machine Learning*.

Welling, M. and Hinton, G. E. (2002). A New Learning Algorithm for Mean Field Boltzmann Machines. *International Joint Conference on Neural Networks*.

White, A. (2006). *A Standard System for Benchmarking in Reinforcement Learning*. M.Sc. Thesis, University of Alberta.