Generalized Prioritized Sweeping

David Andre Nir Friedman Ronald Parr
Computer Science Division, 387 Soda Hall
University of California, Berkeley, CA 94720
{dandre,nir,parr}@cs.berkeley.edu

Abstract

Prioritized sweeping is a *model-based* reinforcement learning method that attempt to focus the agent's limited computational resources to achieve a good estimate of the value of environment states. The classic account of prioritized sweeping uses an explicit, state-based representation of the value, reward, and model parameters. Such a representation is unwieldy for dealing with complex environments and there is growing interest in learning with more compact representations. We claim that classic prioritized sweeping is ill-suited for learning with such representations. To overcome this deficiency, we introduce *generalized prioritized sweeping*, a principled method for generating representation-specific algorithms for model-based reinforcement learning. We then apply this method for several representations, including state-based models and generalized model approximators (such as Bayesian networks). We describe preliminary experiments that compare our approach with classical prioritized sweeping.

1 Introduction

In reinforcement learning, there is a tradeoff between acting in the environment and planning (i.e., performing computations). Model-free methods such as Q-learning [11] take one extreme on this question—the agent does not perform any propagation of learned information. On the other end of the spectrum lie classical dynamic programming methods that reevaluate the utility of every state in the environment after every experiment. Prioritized sweeping (PS) [7] provides a middle ground in that only the most "important" states are updated, according to a priority metric that attempts to measure the anticipated size of the update for each state. Roughly speaking, PS interleaves performing actions in the environment with propagating the values of states. After updating the value of state s, PS examines all states t where the agent might reach s in one step and assigns them priority based on the expected size of the change in their value.

A crucial desideratum for reinforcement learning is the ability to scale-up to complex domains. For this, we need to use *compact* (or *generalizing*) representations of the model and the value function. While it is possible to apply PS in the presence of such representations (e.g., see [1]), we claim that PS is ill-suited in this case. To see this, note that if we use a generalizing model, then after we perform an action a in state s we update our model in a way that may change our estimation of the successors of many states. Thus, we might want to update the value of states that are similar, in some appropriate sense, to s since we have a new estimate of the system dynamics at these states. Note that some of these states might be states that have never been reached before and standard PS will not assign them a priority at all.

In this paper, we present *generalized prioritized sweeping* (GenPS), a method that extends PS to deal with parametric representations for both the model and the value function. If GenPS is used with an explicit, state-space model and value function representation, an algorithm similar to the original (classic) PS results. When a model approximator (such as a *dynamic Bayesian network* [2]) is used, the resulting algorithm prioritizes the states of the environment using the generalizations inherent in the model representation. Thus, states that are rarely if ever visited can be updated based upon the agent's model of their similarity to other states that have been visited.

2 The Basic Principle

We assume the reader is familiar with the basic concepts of MDPs (see, e.g., [6]). We will use the following notation: An MDP is a 4-tuple, $(\mathcal{S}, \mathcal{A}, p, r)$ where $\mathcal{S}$ is a set of *states*, $\mathcal{A}$ is a set of *actions*, $p(t \mid s, a)$ is a *transition model* that captures the probability of reaching state t after we execute action a at state s, and $r()$ is a *reward function* mapping $\mathcal{S}$ into real-valued rewards. In this paper, we focus on infinite-horizon MDPs with a discount factor γ. The aim is to maximize the expected discounted total reward of the agent. Reinforcement learning procedures attempt to achieve this objective when the agent *does not* know p and r. These procedures attempt to find an optimal policy π^* (or, equivalently, a value function V^* or a Q-function Q^*) that maximizes the expected discounted total reward of the agent.

A standard problem in *model-based* reinforcement learning is balancing between planning (i.e., choosing a policy) and execution. Ideally, the agent would compute the optimal value function for its model of the environment each time it updates it. This scheme is unrealistic since finding the optimal policy for a given model is a non-trivial computational task. Fortunately, we can approximate this scheme, if we notice that the approximate model changes only slightly at each step. We can hope that the value function from the previous model can be easily "repaired" to reflect these changes. This approach was pursued in the DYNA [9] framework, where after the execution of an action, the agent updates its model of the environment, and then performs some bounded number of value propagation steps to update its approximation of the value function. Each value-propagation step locally enforces the *Bellman equation* by setting $\hat{V}(s) \leftarrow \max_{a \in \mathcal{A}} \hat{Q}(s, a)$, where $\hat{Q}(s, a) = \hat{r}(s) + \gamma \sum_{s' \in \mathcal{S}} \hat{p}(s' \mid s, a) \hat{V}(s')$, where $\hat{p}(s' \mid s, a)$ and $\hat{r}(s)$ are the agent's approximate model, and $\hat{V}$ is the agent's approximation of the value function.

This raises the question of which states should be updated. In this paper we propose the following general principle:

> **GenPS Principle:** Update states where the approximation of the value function will change the most. That is, update the states with the largest *Bellman error*, $E(s) = |\hat{V}(s) - \max_{a \in \mathcal{A}} \hat{Q}(s, a)|$.

The motivation for this principle is fairly straightforward. The maximum Bellman error can be used to bound the maximum difference between the current value function, $\hat{V}(s)$ and the optimal value function, $V^*(s)$ [12]. This difference bounds the *policy loss*, the difference between the expected discounted reward received under the policy implied by the current value function and the expected discounted reward received under the optimal policy.

To carry out this principle we have to recognize when the Bellman error at a state changes. This can happen at two different stages. First, after the agent updates its model of the world, new discrepancies between $\hat{V}(s)$ and $\max_a \hat{Q}(s, a)$ might be introduced, which can increase the Bellman error at s. Second, after the agent performs some value propagations, $\hat{V}$ is changed, which may introduce new discrepancies.

We assume that an agent is maintaining a value function and a model that are parameterized by θ_V and θ_M. (We will sometimes refer to the vector that concatenates these vectors together into a single, larger vector simply as θ.) For example, an agent using a Bayesian network representation for a transition model would store the conditional probabilities for the network in θ_M (see Section 4).

When the agent observes a transition from state s to s' under action a, the agent updates its environment model by adjusting some of the parameters in θ_M. When performing value-propagations, the agent updates $\hat{V}$ by updating parameters in θ_V. A change in any of these parameters may change the Bellman error at other states in the model. We want to recognize these states without explicitly computing the Bellman error at each one. Formally, we wish to estimate the change in error, $|\Delta_{E(s)}|$, due to the most recent change Δ_θ in the parameters.

We can approximate $|\Delta_{E(s)}|$ by using the gradient of the right hand side of the Bellman equation (i.e. $\max_a \hat{Q}(s, a)$). Thus, we have: $|\Delta_{E(s)}| \approx |\nabla \max_a \hat{Q}(s, a) \cdot \Delta_\theta|$ which estimates the change in the Bellman error at state s as a function of the change in $\hat{Q}(s, a)$. The above still requires us to differentiate over a max, which is not differentiable. There are two ways of overcoming this. First, we could assume that the optimal action $\hat{a}$ remains the same after the change, and use $\nabla \hat{Q}(s, \hat{a})$ to approximate $\nabla \max_a \hat{Q}(s, a)$. This runs the risk of underestimating the change when the change in the value function results in a different action being optimal. In general, we prefer to overestimate the change, since that ensures that value-propagation will eventually be performed for every state. Hence, we use the following approximation:

$$\left| \nabla (\max_a \hat{Q}(s, a)) \cdot \Delta_\theta \right| \leq \max_a \left| \nabla \hat{Q}(s, a) \cdot \Delta_\theta \right| .^1$$

We now define the generalized prioritized sweeping procedure. The procedure maintains a priority queue that assigns to each state s a priority, $pri(s)$. After making some changes, we can reassign priorities by computing an approximation of the change in the value function. To do so, we need a procedure that updates the priorities in the queue. Ideally, this procedure works as follows:

procedure update-priorities (Δ_θ)
  for all $s \in \mathcal{S}$ $pri(s) \leftarrow pri(s) + \max_a |\nabla \hat{Q}(s,a) \cdot \Delta_\theta|$.

Note that when the above procedure updates the priority for a state that has an existing priority, the priorities are added together. In contrast, classic prioritized sweeping uses the maximum of the two priorities as the new priority. The additive method ensures that the priority being kept is an overestimate of the priority of each state. Also, in practice we would not want to reconsider the priority of all states after an update. (We return to this issue below.) Using this procedure, we can now state the general learning procedure:

procedure GenPS ()
  loop
    perform an action in the environment[2]
    update the model; let Δ_θ be the change in θ
    call update-priorities(Δ_θ)
    while there is available computation time
      let $s^{\max} = \arg\max_s pri(s)$
      perform value-propagation for $\hat{V}(s^{\max})$; let Δ_θ be the change in θ
      call update-priorities(Δ_θ)
      $pri(s^{\max}) \leftarrow |\hat{V}(s^{\max}) - \max_a \hat{Q}(s^{\max}, a)|$ [3]

Of course, this abstract description specifies neither how to update the model, nor how to update the value function in the value-propagation steps. Both of these depend on the choices made in the corresponding representation of the model and the value function. Moreover, it is clear that in problems that involve a large state space, we cannot afford to recompute the priority of every state in update-priorities. However, we can simplify this computation by exploiting sparseness in the model and in the worst case we may resort to approximate methods for finding the states that receive high priority after each change.

3 Explicit, State-based Representation

In this section we briefly describe the instantiation of the generalized procedure when the rewards, values, and transition probabilities are explicitly modeled using lookup tables. In this representation, for each state s, we store the expected reward at s, denoted by $\theta_{\hat{r}(s)}$, the estimated value at s, denoted by $\theta_{\hat{V}(t)}$, and for each action a and state t the number of times the execution of a at s lead to state t, denoted $N_{s,a,t}$. From these counts we can reconstruct the transition probabilities $\hat{p}(t \mid s,a) = \frac{N_{s,a,t}+N^0_{s,a,t}}{\sum_{t'} N_{s,a,t'}+N^0_{s,a,t'}}$, where $N^0_{s,a,t}$ are *fictional counts* that capture our prior information about the system's dynamics. Roughly speaking, the magnitude of these counts signifies our confidence in these prior estimates.[4] After each step in the world, these reward and probability parameters are updated in the straightforward manner. Value propagation steps in this representation assign $\theta_{\hat{V}(t)}$ the right hand side of the Bellman equation.

To apply the GenPS procedure we need to derive the gradient of the Bellman equation for two situations: (a) after a single step in the environment, and (b) after a value update. It is easy to verify that the partial derivatives are:

$$\frac{\partial \hat{Q}(s,a)}{\partial \theta_{\hat{r}(t)}} = 1 \text{ if } t = s, 0 \text{ otherwise}$$

[1]We might replace the scope of the max operator to handle only "reasonable" choices of actions. This can be done in a straightforward manner, and we defer the details for now.

[2]There is an issue of which action to perform. Since this issue is orthogonal to the our main topic, we do not elaborate on it here. We refer the interested reader to [6, 7, 9].

[3]In general, this will assign the state a new priority of 0, unless there is a self loop. In this case it will easy to compute the new Bellman error as a by-product of the value propagation step.

[4]Formally, we are using multinomial *Dirichlet* priors. See, for example, [5] for an introduction to these Bayesian methods.

$$\frac{\partial \hat{Q}(s,a)}{\partial N_{s',a',t'}} = \frac{\gamma}{\sum_t N_{s,a,t}+N^0_{s,a,t}} \left(V(t') - \sum_t \hat{p}(t \mid s,a)V(t)\right) \text{ if } a' = a \text{ and } s = s', 0 \text{ otherwise}$$
$$\frac{\partial \hat{Q}(s,a)}{\partial \theta_{\hat{V}(t)}} = \gamma \hat{p}(t \mid s,a)$$

Using these derivatives we can examine which states change their priorities when we change the model. As noted above there are two cases. In case (a), the model changes after performing action $s \xrightarrow{a} t$. This results in changing the parameters $\theta_{\hat{r}(s)}$ and $N_{s,a,t}$. It is easy to verify that in this case $\nabla Q(s',a') \cdot \Delta_\theta$ is nonzero only if $s' = s$ and $a' = a$. In case (b), the value function changes after updating the value of a state s. This results in changing the parameter $\theta_{\hat{V}(s)}$. It is easy to verify that in this case $\nabla Q(s',a') \cdot \Delta_\theta$ is nonzero only if $\hat{p}(s \mid s',a') > 0$, i.e., s is reachable from s'. In both cases, it is straightforward to locate the states where the Bellman error might have have changed, and the computation of the new priority is more efficient than computing the Bellman-error.[5]

Now we can relate GenPS to standard prioritized sweeping. The PS procedure has the general form of this application of GenPS with three minor differences. First, after performing a transition $s \xrightarrow{a} t$ in the environment, PS immediately performs a value propagation for state s. As explained above, when using a state-based representation, this is indeed the only state for which the Bellman error might change after updating the model. Second, after performing a value propagation for state t, PS updates the priority of states s that can reach t with the value $\max_a \hat{p}(t \mid s,a) \cdot \Delta_{\hat{V}(t)}$. The priority assigned by GenPS is the same quantity multiplied by γ. Since PS does not introduce priorities after model changes, this multiplicative constant does not change the order of states in the queue. Thirdly, as noted above, GenPS uses addition to combine the old priority of a state with a new one, while PS uses max.

This discussion shows that PS can be thought of as a special case of GenPS when the agent uses explicit state-based representation. As we show in the next section, when the agent uses more compact representations, we get a procedure where the prioritization strategy is quite different than that used in PS. Thus, we claim that PS implicitly assumes that the agent uses an explicit state-based model.

4 Factored Representation

We now examine a compact representation of $\hat{p}(s' \mid s,a)$ that is based on *dynamic Bayesian networks* (DBNs). DBNs have been combined with reinforcement learning before in [10], where they were used primarily as a means of reducing the effort required to learn an accurate model. We will show that they can also be used with prioritized sweeping to focus the agent's attention on groups of states that are affected as the agent refines its environment model.

We start by assuming that the environment state is described by a set of attributes. Let $X_1,\ldots,X_n$ be *random variables* that describe the values of these attributes. For now, we assume that each variable can take values from from a finite set $\mathit{Val}(X_i)$. An *assignment* of values $x_1,\ldots,x_n$ to these variables describes a particular environment state. Similarly, we assume that the agent's action is described by random variables $A_1,\ldots,A_k$. To model the system dynamics, we have to represent the probability of transitions $s \xrightarrow{a} t$, where s and t are two assignments to $X_1,\ldots,X_n$ and a is an assignment to $A_1,\ldots,A_k$. To simplify the discussion, we denote by $Y_1,\ldots,Y_n$ the agent's state after the action is executed (e.g., the state t). Thus, $p(t \mid s,a)$ is represented as a conditional probability $P(Y_1,\ldots,Y_n \mid X_1,\ldots,X_n,A_1,\ldots,A_k)$.

A DBN model for such a conditional distribution consists of two components. The first is a directed acyclic graph where each vertex is labeled by a random variable and in which the vertices labeled $X_1,\ldots,X_n$ and $A_1,\ldots,A_k$ are roots. This graph specifies the *factorization* of the conditional distribution:

$$P(Y_1,\ldots,Y_n \mid X_1,\ldots,X_n,A_1,\ldots,A_k) = \prod_{i=1}^{n} P(Y_i \mid \mathit{Pa}_i), \tag{1}$$

where Pa_i are the parents of Y_i in the graph. The second component of the DBN model is a description of the conditional probabilities $P(Y_i \mid \mathit{Pa}_i)$. Together, these two components describe a unique conditional distribution. In the case of discrete variables, the simplest representation of $P(Y_i \mid \mathit{Pa}_i)$ is by a table that contains a parameter $\theta_{i,y,z} = P(Y_i = y \mid \mathit{Pa}_i = z)$ for each possible combination of $y \in \mathit{Val}(Y_i)$ and $z \in \mathit{Val}(Pa_i)$ (note that z is a joint assignment to several random variables). It is easy to see that the "density" of the DBN graph determines the number of parameters needed. In particular, a *complete* graph, to which we cannot add an arc without violating the constraints,

[5] Although $\frac{\partial \hat{Q}(s,a)}{\partial N_{s,a,t}}$ involves a summation over all states, it can be computed efficiently. To see this, note that the summation is essentially the old value of $Q(s,a)$ (minus the immediate reward) which can be retained in memory.

is equivalent to a state-based representation in terms of the number of parameters needed. On the other hand, a sparse graph requires few parameters.

In this paper we assume that the learner is supplied with the DBN structure and only has to learn the conditional probability entries. This form of knowledge is often easy to assess from an expert. We discuss how to relax this assumption in Section 6. There several possible ways of learning the parameters for the DBN from experience. We refer the interested reader to [5, 8]. As in the state-based representation, we use Dirichlet priors for each multinomial distribution. In this method, we assess the conditional probability $\theta_{i,y,z}$ using prior knowledge and the frequency of transitions observed in the past where $Y_i = y$ among these transitions where $Pa_i = z$. Learning amounts to keeping counts $N_{i,y,z}$ that record the number of transitions where $Y_i = y$ and $Pa_i = z$ for each variable Y_i and values $y \in Val(Y_i)$ and $z \in Val(Pa_i)$. Our prior knowledge is represented by fictional counts $N^0_{i,y,z}$. The we estimate probabilities using the formula $\theta_{i,y,z} = \frac{N_{i,y,z}+N^0_{i,y,z}}{N_{i,\cdot,z}}$, where $N_{i,\cdot,z} = \sum_{y'} N_{i,y',z} + N^0_{i,y',z}$.

Given this machinery, we can understand why GenPS should act differently when using such generalizing models. Consider a simple example: Suppose that an agent is traveling in a maze world where the environment state is described by two attributes: the position of the agent, and whether agent is carrying a key to a gold chest (and the agent's task is to retrieve the gold). In this example we assume that the agent employs a DBN whose structure implies that the agent does not loose the key when it moves. Now suppose that the agent learns that at the state $V(pos_{gold}, \textit{have-key})$ it receives a high reward. Now suppose that our agent is at another position pos_0 without the key, it performs the action *move-forward* and arrives at pos_{gold}. This causes the agent to update its model. In particular, the agent learns that performing *move-forward* from position pos_0 might lead to position pos_{gold} *regardless* of whether it has the key. Thus, the agent learns that from state $(P_0, \textit{have-key})$ it might reach the desirable state $(P_{gold}, \textit{have-key})$. This is useful information that should cause the agent to change the value of $(P_0, \textit{have-key})$. However, it is easy to see that PS would *not* change the priority of the state $(P_0, \textit{have-key})$ since it is not a direct predecessor of $(P_0, \neg\textit{have-key})$. On the other hand, as we shall see, when we derive GenPS for the DBN representation, we get a procedure that does increase the priority of $(P_0, \textit{have-key})$.

We now identify which states should be reconsidered after we update the DBN parameters. Recall that this requires estimating the term $\nabla Q(s,a) \cdot \Delta_\theta$. We start by noting that Δ_θ is sparse, since exactly one count of the form $N_{i,y,z}$ changes for each i and this count is incremented by 1. Thus, after making the transition $s^* \xrightarrow{a^*} t^*$, we have that

$$\nabla Q(s,a) \cdot \Delta_\theta = \sum_i \frac{\partial Q(s,a)}{\partial N_{i,y_i^*,z_i^*}}, \tag{2}$$

where y_i^* and z_i^* are the assignments to Y_i and Pa_i, respectively, in $s^* \xrightarrow{a^*} t^*$. (Recall that s^*, a^* and t^* jointly assign values to all the variables in the DBN.)

To compactly describe $\frac{\partial Q(s,a)}{\partial N_{i,y_i^*,z_i^*}}$ we introduce some notation. We say that a transition $s \xrightarrow{a} t$ is *consistent* with an assignment $X = x$ for a vector of random variables X, denoted $(s,a,t) \models (X = x)$, if X is assigned the value x in $s \xrightarrow{a} t$. We also need a similar notion for a partial description of a transition. We say that s and a are consistent with $X = x$, denoted $(s,a,\cdot) \models (X = x)$, if there is a t such that $(s,a,t) \models (X = x)$.

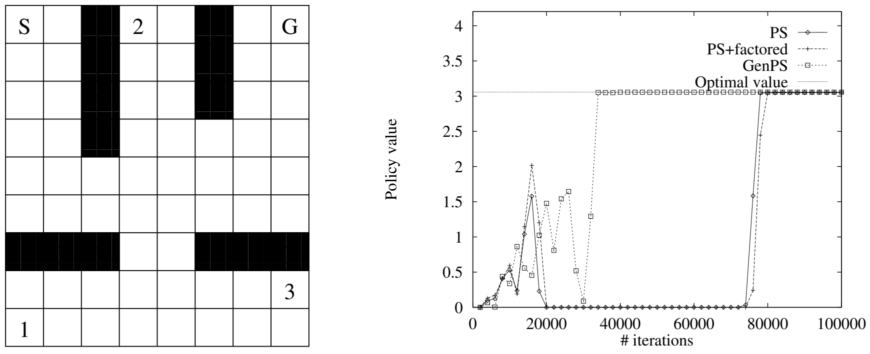
Using this notation, we can show that if s, a are consistent with $Pa_i = z_i^*$, then

$$\frac{\partial Q(s,a)}{\partial N_{i,y_i^*,z_i^*}} = \frac{\gamma}{N_{i,\cdot,z_i^*}} \left[\frac{1}{\theta_{i,y_i^*,z_i^*}} \sum_{t:(s,a,t)\models y_i^*,z_i^*} \hat{p}(t \mid s,a)\hat{V}(t) - \sum_{t:(s,a,t)\models z_i^*} \hat{p}(t \mid s,a)\hat{V}(t) \right],$$

and if s, a are inconsistent with $Pa_i = z_i^*$, then $\frac{\partial Q(s,a)}{\partial N_{i,y_i^*,z_i^*}} = 0$.

This expression shows that if s is similar to s^* in that both agree on the values they assign to the parents of some Y_i (i.e., (s, a^*) is consistent with z_i^*), then the priority of s would change after we update the model. The magnitude of the priority change depends upon both the similarity of s and s^* (i.e. how many of the terms in (2) will be non-zero), and the value of the states that can be reached from s.

The evaluation of $\frac{\partial Q(s,a)}{\partial N_{i,y_i^*,z_i^*}}$ requires us to sum over a subset of the states –namely, those states t that are consistent with z_i^*. Unfortunately, in many cases this will be a large fragment of the state space. If the number of environment states is not large, then this might be a reasonable cost to pay for the additional benefits of GenPS. However, this might be a serious problem when we have a large state space, which are the cases where we expect to gain the most benefit from using generalized representations such as DBN.



(a) (b)

Figure 1: (a) The maze used in the experiment. S marks the start space, G the goal state, and 1, 2 and 3 are the three flags the agent has to set to receive the reward. (b) A graph showing the performance of the three procedures on this example. PS is GenPS with stated-based model, PS+factored is the same procedure but with a factored model, and GenPS exploits the factored model in prioritization. Each curve is the average of 5 runs.

In these situations, we propose a heuristic approach for estimating $\nabla Q(s,a)\Delta_\theta$ without summing over large numbers of states for computing the change of priority for each possible state. This can be done by finding upper bounds on $\frac{\partial Q(s,a)}{\partial N_{i,y_i^*,z_i^*}}$. There are many approaches for doing this that will involve different tradeoffs between accuracy and speed. A simple approach is to overestimate the probabilities $\hat{p}(s' \mid s,a)$ inside the summations in $\frac{\partial Q(s,a)}{\partial N_{i,y_i^*,z_i^*}}$. (We defer the technical details of this computation to the full version of the paper.) Once we have computed these upper bounds, we estimate the priority change for each state s as follows. We use the notation $(s,a) \sim_i (s^*,a^*)$ if s and a agree with s^* and a^* on the assignment Pa_i. Similarly, $s \sim_i s^*$ if they both agree on the assignment to Pa_i. If we compute C_i such that $\left|\frac{\partial Q(s,a)}{\partial N_{i,y_i^*,z_i^*}}\right| \leq C_i$, we have that

$$|\nabla Q(s,a)\Delta_{\theta_M}| \leq \textstyle\sum_{i:(s,a)\sim_i(s^*,a^*)} C_i \;\leq \sum_{i:(s,a^*)\sim_i(s^*,a^*)} C_i \;= \sum_{i:s\sim_i s^*} C_i.$$

Thus, to evaluate the priority of state s, we simply find how "similar" it is to s^*. Note that it is relatively straightforward to use this equation to enumerate all the states where the priority change might be large.

Finally, we note that the use of a DBN as a model does not change the way we update priorities after a value propagation step. If we use an explicit table of values, then we would update priorities as in the previous section. If we use a compact description of the value function, then we can apply GenPS to get the appropriate update rule.

5 An Experiment

We conducted an experiment to evaluate the effect of using GenPS with a generalizing model. We use a maze domain similar to the one described in [7, p.123]. The maze, shown in Figure 1(a), contains 59 cells, and 3 binary flags, resulting in $59 \times 2^3 = 472$ possible states. Initially the agent is at the start state (marked by S) and the flags are reset. The agent has four possible actions, up, down, left, and right, that succeed 80% of the time, and 20% of the time the agent moves in an unintended perpendicular direction. The i'th flag is set when the agent leaves the position marked by i. The agent receives a reward when it arrives at state G and all of the flags are set. In this situation all of the agent's actions reset the game.

As noted in [7], this environment exhibits independencies. Namely, the probability of transition from one position to another does not depend on the flag settings. These independencies can be captured easily by a simple DBN. Our experiment is designed to test to what extent GenPS exploits the knowledge of these independencies for faster learning.

We tested three procedures. The first is GenPS, which uses an explicit state-based model. As explained above, this variant is essentially PS. The second procedure uses a factored model of the environment for learning the model parameters, but uses the same prioritization strategy as the first one. The third procedure uses the GenPS prioritization strategy we describe in Section 4. All three procedures use the exploration strategy described in [7] with $t_{bored} = 5$. Finally, in each iteration these procedures process at most 10 states from the priority queue.

The results are shown in Figure 1(b). As we can see, the GenPS procedure converged faster than the procedures that used classic PS prioritization. As explained above, our hypothesis is that this procedure explioted regularties in the domain to better propogate the value function, and thus, convereged faster to the optimal policy. Note that since we are using the exploration strategy of [7], states for which the procedure performed few value propogations are always considered more attractive. Thus, the PS procedure converged to the optimal policy only after it "visited" each of these states. We are currently performing experiments with other exploration strategies that do not directly modify the initial value function.

6 Discussion

We have presented a general method for approximating the optimal use of computational resources during reinforcement learning. Like classic prioritized sweeping, our method aims to perform only the most beneficial value propagations. By using the gradient of the Bellman equation our method generalizes the underlying principle in prioritized sweeping. The generalized procedure can then be applied not only in the explicit, state-based case, but in cases where approximators are used for the model. In the complete version of this paper we also show to apply GenPS to function approximators, such as averagers [4]). We are working on applying GenPS to other types of model and function approximators.

The focus of this paper was on interleaving planning with execution during learning. Of course, there are many other issues that need to be dealt with in model-based reinforcement learning. For example, in Section 4, we assumed that the agent is given the DBN structure. In many applications, this might be an unreasonable assumption. We are interested in combining our approach with recent results on incremental learning of Bayesian networks [3].

We are optimistic that generalized prioritized sweeping will extend the well-known advantages of prioritized sweeping to a much broader and general class of representations. We believe that the generalized procedure may amplify the benefits of function approximation by exploiting the structure of the approximator in every step of the planning process and allowing the agent to maintain an accurate value function for parts of the environment that are rarely or never visited.

Generalized prioritized sweeping allows the agent to use a model to allocate planning or replanning resources across a wide variety of representation methods and problem domains. This elevates the status of the model from a simple tabulation of the agent's experiences to a powerful means of directing the agent's computational processes. This benefit is amplified by the fact that many representations, such as DBNs, provide a principled means of representing and transferring prior knowledge. Thus, generalized prioritized sweeping underscores the importance of two forms of knowledge, knowledge that is given to the agent in the structure of the initial model and value function representations, and knowledge that is acquired by the agent in the form of an environment model.

References

[1] S. Davies. Multidimensional triangulation and interpolation for reinforcement learning. In *Advances in Neural Information Processing Systems 9*. 1996.

[2] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

[3] N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *Proc. 13th Conf. on Uncertainty in Artificial Intelligence*. 1997. To appear.

[4] G. J. Gordon. Stable function approximation in dynamic programming. In *Proc. 12th Int. Conf. on Machine Learning*, 1995.

[5] D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995. Revised November 1996.

[6] L. P. Kaelbling, M. L. Littman and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[7] A. W. Moore and C. G. Atkeson. Prioritized sweeping—reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[8] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 1995.

[9] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning: Proc. 7th Int. Conf.*, 1990.

[10] P. Tadepalli and D. Ok. Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *Proc. 13th Int. Conf. on Machine Learning*, 1996.

[11] C. J. Watkins. *Models of Delayed Reinforcement Learning*. PhD thesis, Psychology Department, Cambridge University. 1989.

[12] R. J. Williams and L. C. III Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, College of Computer Science, Northeastern University. 1993.