Lightweight Adaptation in Model-Based Reinforcement Learning

Lisa Torrey
St. Lawrence University
ltorrey@stlawu.edu

Abstract

Reinforcement learning algorithms can train an agent to operate successfully in a stationary environment. Most real-world environments, however, are subject to change over time. Research in the areas of transfer learning and lifelong learning addresses this problem by developing new algorithms that allow agents to adapt to environment change. Current trends in this area include model-free learning and data-driven adaptation methods. This paper explores in the opposite direction of those trends. Arguing that model-based algorithms may be better suited to the problem, it looks at adaptation in the context of model-based learning. Noting that standard algorithms themselves have some built-in capability for adaptation, it analyzes when and why a standard algorithm struggles to adapt to environment change. Then it experiments with lightweight and straightforward methods for adapting effectively.

Introduction

In reinforcement learning (RL), an agent explores an environment, and its actions change the state of that environment. Different action sequences cause different state sequences and also earn different rewards. By accumulating knowledge through experience, an agent learns how to maximize its rewards.

RL algorithms are optimized for stationary environments. Most realistic environments, however, are subject to change over time. Research in the areas of transfer learning and lifelong learning addresses this problem by developing new algorithms that allow agents to adapt to environment change.

An RL environment is characterized by its set of possible states, the set of possible actions, the state transitions caused by actions, and the rewards earned by actions. This paper focuses on types of environment change that cause some actions to have different effects or earn different rewards. When this occurs, a trained agent's knowledge becomes at least partially incorrect, but often still contains useful information.

Reinforcement learning is most often done in a *model-free* fashion. In this type of RL, agent knowledge is limited to a *policy*, which maps environment states directly to desirable actions. Another type of RL, called *model-based*, has agents accumulate both a policy and an *environment model*, which stores knowledge about the immediate effects of actions.

Model-free RL is effective at training agents to behave successfully, but it produces a shallow kind of learning that does not include a real understanding of the environment. This limits the ability of model-free agents to detect environment change and adapt to it. Model-based RL produces a deeper kind of learning that includes direct knowledge about the environment. This can allow a model-based agent to detect changes when they occur, and also provides useful information for adapting. This paper therefore focuses on adaptation in model-based RL.

Some basic RL algorithms have theoretical guarantees of success under certain conditions, one of which is a stationary environment (Sutton and Barto 1998). However, theory and practice often have a tenuous relationship in reinforcement learning. In practice, RL agents can often adapt successfully to environment changes if they are infrequent enough or slow enough. The incremental learning process of RL provides some built-in capability for adaptation.

This paper takes a well-known model-based RL algorithm (prioritized sweeping) and explores its capabilities and limitations with respect to environment change. Experiments in a simple domain show that adaptation can be possible, but that some types of changes can cause it to be quite slow. An analysis of these situations leads to experiments with lightweight and straightforward extensions to prioritized sweeping that can achieve effective adaptation.

Related Work

Environment change in reinforcement learning has been studied under several names, including *transfer learning* and *lifelong learning*. Transfer learning involves one or more *source tasks* that are followed by a *target task*. Lifelong learning takes place over an extended agent lifetime, and involves a sequence of tasks.

A large body of work in RL transfer is summarized in a recent survey (Taylor and Stone 2009). Three model-based approaches seem most relevant here. The first uses the old environment to produce prior probabilities for a Bayesian learner in the new environment (Sunmola and Wyatt 2006). The second uses experience samples from an old environment to make better predictions earlier in a new environment (Taylor, Jong, and Stone 2008). The third also transfers

experience samples, but focuses on screening the samples for applicability in the new environment (Lazaric, Restelli, and Bonarini 2008).

In lifelong RL, some work focuses on allowing agents to develop and maintain multiple capabilities. For example, the EVM architecture has an ensemble of parallel learning cells that can specialize independently for different environments (Nowostawski 2009). Another approach uses hierarchical RL to split learning into multiple levels, providing some stability in the event of change (Kleiner, Dietl, and Nebel 2002).

Other work in lifelong RL focuses on the problem of transitioning between environments. Some algorithms approach this problem by developing policy-learning biases based on knowledge from the old environment (Thrun and Mitchell 1995; Tanaka and Yamamura 1997). Along similar conceptual lines is a method that uses statistics from old environments to influence which regions in a new environment receive the most attention (Tanaka and Yamamura 2003). Another approach is to make direct use of the policy from the old environment, but to relearn parts of it that fail in the new one (Minato and Asada 1998).

This paper differs from much of its related work by focusing on model-based RL, since most work in this area uses model-free learning. Those studies that do use model-based RL tend to propose data-driven algorithms for adaptation. This paper focuses instead on more lightweight methods, inspired by an analysis of when standard RL algorithms struggle to adapt to environment change. These methods attempt to leverage model-based knowledge in common-sense ways.

Experimental Domain

An ideal domain for the goals of this paper would be conceptually simple, yet contain moderately difficult learning tasks and a variety of ways to introduce change. The domain should be simple to facilitate a close analysis of algorithm behavior, but the tasks should be difficult enough that effective adaptation can make a significant difference. This section introduces a domain called Mail that is designed to meet these requirements.

In the Mail domain, the agent is a courier whose job is to move packages between rooms. Rooms are laid out in a grid, and each room initially contains one package. Each package has an unknown destination room. The agent earns rewards by delivering packages to their destinations.

Tasks in the Mail domain are episodic. In each episode, the agent has a fixed number of steps to act. At each step, it chooses among these actions: *move north*, *move east*, *move south*, *move west*, and *exchange*.

Movement actions have obvious results, except that attempts to move beyond the grid of rooms have no effect. The *exchange* action puts down the package the agent is carrying (if any) and picks up the other package in the agent's current room (if any). When a package is put down in its destination room, the agent receives a reward and the package disappears from the environment. Any action may, with a small probability, fail to have any effect at all; this makes the domain mildly non-deterministic.

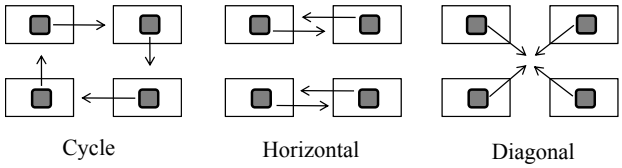

Figure 1: Three initial environment configurations in the Mail domain, with package destinations shown by arrows.

Tasks in this domain can be made arbitrarily challenging simply through combinatorics. The number of possible environment states increases steeply with the grid size. Figure 1 shows three environments in the Mail domain that are small, yet non-trivial to learn. Each of these tasks allows 20 steps per episode and provides a reward of 10 for each package delivery.

Model-Based Reinforcement Learning

A model-based RL agent begins by observing the state s of its environment and choosing an action a to perform. Then it observes the resulting state s' and receives a reward r for its action. It uses this information to update its knowledge, which consists of both an environment model and a policy.

Let S represent the set of possible states of the environment and let A represent the set of possible actions. Each environment has a function $P(s, a, s')$, which gives the probability that taking action a in state s causes a transition to state s', and a function $R(s, a, s')$, which specifies the reward earned by this transition. In discrete domains, an environment model is a pair of tables that approximate these functions. Learning an environment model then means updating table entries based on observations.

A common way to represent a policy is with a *Q-function* $Q(s, a)$, which estimates the total reward the agent can earn starting in state s and taking action a. Given an accurate Q-function, the agent can maximize its rewards by choosing actions with maximal Q-values. Learning a policy then means updating Q-values to make them more accurate. Given approximate P and R, estimates for Q_{sa} can be computed according to the function $estimate(s, a)$ below. The *discount* parameter $\gamma \in (0, 1]$ weights the importance of immediate rewards relative to more distant rewards.

$$estimate(s, a) = \sum_{s'} P_{sas'} \left[R_{sas'} + \gamma \text{max}_{a'} Q_{s'a'} \right]$$

Even before the Q-function is accurate, an RL agent typically chooses actions with maximal Q-values most of the time. This produces nearly random behavior at first, when the Q-values are uniformly low, but over time the agent discovers which actions lead to better rewards. To avoid settling for suboptimal behavior, an agent needs to perform occasional *exploration* actions throughout the learning process. This is often accomplished via ϵ-greedy exploration: with probability $\epsilon \in (0, 1)$, an agent chooses a random action instead of the one with the highest Q-value.

One classic approach to model-based RL is the *Dyna* architecture (Sutton 1990). After each observation in the environment, a Dyna agent updates P and R and then selects

Table 1: The prioritized sweeping algorithm, based on pseudocode from Sutton and Barto (1998).

Initialize empty tables P, R, and Q
Initialize empty priority queue $PQueue$
Learning procedure:
  Observe current state s
  Choose action $a \leftarrow \text{argmax}_{\bar{a}} Q_{s\bar{a}}$
  With probability ϵ, change a to a random action
  Execute action a and observe next state s' and reward r
  Update $P_{sas'}$ and $R_{sas'}$
  Compute priority $p \leftarrow |estimate(s,a) - Q_{sa}|$
  If $p > \theta$, insert (s,a) onto $PQueue$ with priority p
  Repeat N times while $PQueue$ is non-empty:
    $(s,a) \leftarrow$ remove highest-priority item from $PQueue$
    Update $Q_{sa} \leftarrow Q_{sa} + \alpha\,(estimate(s,a) - Q_{sa})$
    For each $(\bar{s},\bar{a})$ for which $P_{\bar{s}\bar{a}s} > 0$:
      Compute priority $\bar{p} \leftarrow |estimate(\bar{s},\bar{a}) - Q_{\bar{s}\bar{a}}|$
      If $\bar{p} > \theta$, insert $(\bar{s},\bar{a})$ onto $PQueue$ with priority $\bar{p}$

N random Q-values to update. The *prioritized sweeping* algorithm is a variant of Dyna in which the agent focuses on larger updates, rejecting any smaller than a threshold θ, and prioritizes them in a way that tends to propagate changes quickly (Moore and Atkeson 1993).

Prioritized sweeping is described in Table 1. The *learning rate* parameter $\alpha \in (0, 1]$ controls how quickly the Q-values change.

Prioritized Sweeping and Change

This section explores how prioritized sweeping copes with environment changes through experiments in the Mail domain. Each change experiment is repeated three times with three independent tasks: the cycle, horizontal, and diagonal tasks from Figure 1. Agents learn until their performance stabilizes in these initial environments, and then they continue learning in the corresponding changed environments.

Learning curves are shown for all three tasks, starting at the time of the change. Curves representing these agents are labeled *old-model*, since they proceed with the complete model from the old environment. After each training episode, performance is estimated by averaging rewards over 100 episodes in which no learning or exploration occurs. To generate a smooth final curve for an experiment, 100 independent curves are averaged together.

For comparison, there are also curves labeled *fresh-start*. These represent agents that start learning just as the environment changes, without any old knowledge. They serve as a baseline; agents are not considered to be adapting effectively unless they at least keep up with these curves. Learning parameters from Table 1 are always set to optimize the fresh-start curves, and typical values are $\alpha = 0.5, \epsilon = 0.1, \gamma = 0.95, \theta = 0.1, N = 10$.

Very simple changes are sufficient to establish the limitations of adaptation with prioritized sweeping. Consider first a *larger-reward* change, in which the rewards for package deliveries increase from 10 to 15. This scenario should be trivial for adaptation, because it requires no change in agent behavior, and Figure 2 shows that this is indeed the case.

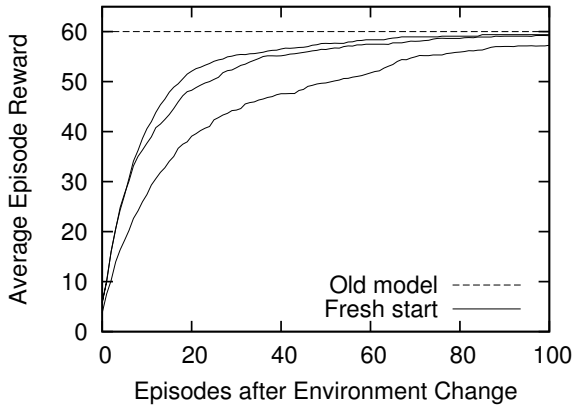

Figure 2: The *larger-reward* change. Prioritized sweeping adapts with no loss in performance.

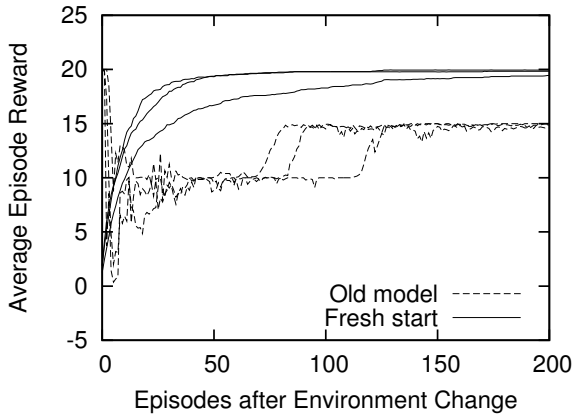

Figure 3: The *smaller-reward* change. Prioritized sweeping struggles to adapt, and agents would be better off starting fresh.

The old-model agent can perform perfectly right away, suffering no decrease in rewards as it adjusts Q-values to their new levels.

Note that the cycle, horizontal, and diagonal environments have separate curves in all figures, but they are not individually labeled, because they are all qualitatively similar when generated by the same learning algorithm. Sometimes they coincide completely, as with the old-model curve in Figure 2. The figure keys are therefore simplified to distinguish only between different learning algorithms.

Next consider a *smaller-reward* change, in which the rewards for package deliveries decrease from 10 to 5. This scenario also requires no change in agent behavior, so it seems trivial as well. However, Figure 3 shows that prioritized sweeping handles it quite poorly. While it begins with optimal performance, it crashes as soon as it attempts to adjust its knowledge to reflect the change. The fresh-start agent actually learns the new environment faster than the old-model agent can adapt.

To understand why, imagine what happens when the agent makes its first delivery in the new environment. The re-

ward is smaller than it expects, so it updates its environment model and decreases some Q-values towards their new levels. The Q-value decreases propagate along the path the agent took, even though it is still the correct path. Now other source-task paths to the same final state look better, though they are not. It takes many episodes to correct the Q-values along all possible paths, and during that time the agent performs poorly.

This problem will occur with any environment change in which an expected reward is reduced, in any domain with multiple ways to reach the expected reward. This describes most environment changes and most real-world domains. Prioritized sweeping unlearns expectations slowly, and multiple paths compound the problem. Though optimistic Q-value initialization is used effectively in some RL algorithms (Szita and Lorincz 2008), optimistic expectations appear to be harmful to adaptation.

Path Transfer

Humans adapting to the simple changes above would not have the same difficulty. After observing the change in the primary path, we would not need to try the others. Prioritized sweeping could simulate this by removing unnecessary alternate paths, allowing the agent to focus only on the recommended path.

As Table 2 describes, this can be done by performing one procedure when an environment change occurs. The costs of transfer learning are typically measured in episodes rather than computation, since cognition can be considered cheap compared to experience. However, it is worth noting that this algorithm is not computationally expensive either, since it requires only a one-time sweep through an agent's environment model.

Learning curves for agents using this algorithm are labeled *path-transfer*. With this approach, consider again the smaller-reward change that caused difficulties earlier. Figure 4 shows that adaptation can now proceed quickly. Agents suffer a small dip in performance as they correct their knowledge, but remain well ahead of the fresh-start baseline.

Consider also a *pickup-penalty* change, in which a penalty of -5 occurs when the agent picks up a package, offset by the usual +10 reward when it delivers. This scenario also represents a decrease in expected rewards, and Figure 5 shows that path transfer handles it similarly well.

Next consider a *broken-action* change: the action for moving east becomes an action for moving diagonally, as

Table 2: Agents using the path-transfer variant of prioritized sweeping follow Table 1, but perform this procedure once when an environment change occurs.

Begin in the episode initial state s
Until reaching the episode final state:
Find the action a with highest Q_{sa}
Find the likely next state s' according to P
Mark the transition (s, a, s')
Let $s \leftarrow s'$
For all unmarked transitions (s, a, s'):
Clear entries in P, R, and Q

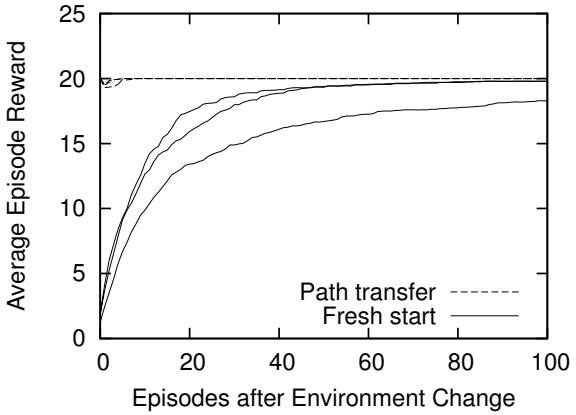

Figure 4: The *smaller-reward* change revisited. With the path-transfer approach, prioritized sweeping adapts effectively.

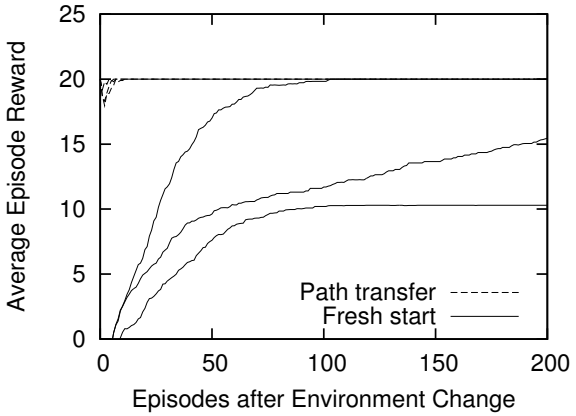

Figure 5: The *pickup-penalty* change. With the path-transfer approach, prioritized sweeping adapts effectively.

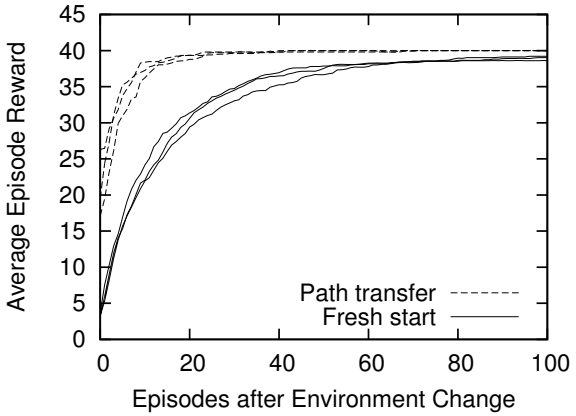

Figure 6: The *broken-action* change. With the path-transfer approach, prioritized sweeping adapts effectively.

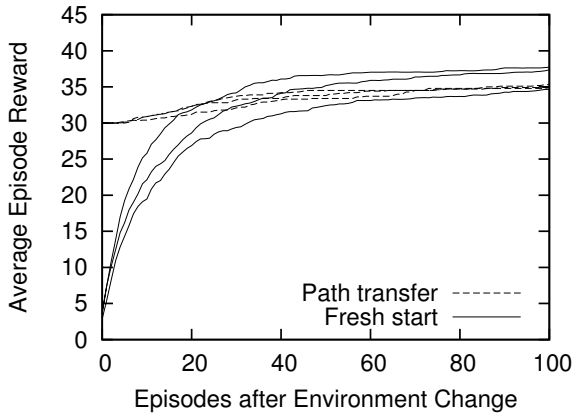

Figure 7: The *new-destination* change. With the path-transfer approach, prioritized sweeping starts out well, but falls behind later due to slow adaptation.

if due to a mechanical malfunction. This scenario does not directly reduce rewards, but it does violate expectations by causing agents to fall off their recommended path whenever they attempt to move east. Figure 6 shows that path transfer can handle this type of scenario too. Normal exploration and learning allow agents to complete the task after they fall off their path, and they remain ahead of the fresh-start baseline.

Note that it should be possible to handle this type of change better than path transfer does. Old knowledge may include ways to get back on the recommended path after falling off. However, this knowledge is removed along with the alternate paths in this approach.

Another type of environment change further illustrates the limitations of path transfer. In the *new-destination* change, the delivery destination of the agent's last package shifts to a different room. In this scenario, the recommended path is actually misleading, and partially distracts the agent from learning the optimal path. Figure 7 shows that path transfer starts out well, but the agents adapt so slowly that it falls behind the fresh-start baseline. The need to unlearn incorrect expectations makes this type of change difficult to handle.

Step Transfer

Humans adapting to the moderate changes above would still not have the same difficulty. We would recall alternate paths when necessary, and we would quickly lose trust in old knowledge when it failed to produce success in a new environment. Prioritized sweeping could simulate this by keeping all the old knowledge accessible and using it for recommendations only as needed, retracting knowledge that seems untrustworthy.

As Table 3 describes, this can be done by inserting three procedures into each iteration of prioritized sweeping. Note that this algorithm does not introduce a large amount of extra computational complexity either. The first two procedures are cheap, and the third, which requires a sweep through an environment model, is only performed once if at all.

Judging knowledge to be untrustworthy requires some discretion, since unexpected results may occur occasionally due to non-determinism. Furthermore, unexpected transitions that still yield expected rewards should be tolerated, since differences may be unimportant until they affect rewards. A recommendation from old knowledge is therefore judged untrustworthy only if it incorrectly predicts both the state and the reward, and has done so enough times to reasonably rule out non-determinism as the cause. In the Mail domain, and probably in most domains, two failures is enough to indicate a bad recommendation.

Table 3: Agents using the step-transfer variant of prioritized sweeping follow Table 1, but insert these procedures into each iteration of prioritized sweeping.

If an environment change has occurred:
Shelve the old P, R, and Q tables
Initialize new empty tables P, R, and Q
Initialize empty recommendation list L
Initialize $trust \leftarrow true$
Before choosing an action in a state s:
If $trust$ is still $true$:
  Get the action a with highest old Q_{sa}
  Get the likely next state s' according to old P
  If recommendation (s, a, s') is not already in L:
    Add (s, a, s') to L
    Copy entries for (s, a, s') from old P, R, Q to new
After a transition $(s, a, \bar{s})$:
If $trust$ is still $true$:
  If a recommendation in L incorrectly predicts $s' \neq \bar{s}$:
    If the recommendation (s, a, s') is untrustworthy:
      Set $trust \leftarrow false$
      Clear entries for (s, a, s') in new P and R
      Decrease all entries in new Q by the old Q_{sa}

When a recommendation has been rejected, Q-values are reduced to remove the false expectations it produced. It is better to reduce Q-values too much than too little in order to avoid false optimism. Furthermore, since old knowledge beyond this recommendation is likely to be invalid, agents using this algorithm disregard old knowledge after rejecting a recommendation.

Learning curves for agents using this algorithm are labeled *step-transfer*. With this approach, consider again the broken-action change that was handled suboptimally earlier. Figure 8 shows that adaptation is now much more effective. Agents are able to access alternative paths to immediately compensate for the broken action.

Also consider again the new-destination change that caused difficulties earlier. Figure 9 shows that adaptation is much improved here as well. Agents recognize that their old knowledge is incorrect about the last delivery, retract the harmful expectations, and complete the task through normal exploration and learning.

This approach should be mostly resistant to falling behind the fresh-start baseline, because it essentially reverts to standard prioritized sweeping as soon as it encounters any significant differences between the tasks. However, there are two types of scenarios where step transfer might behave undesirably. One is if the optimal path in the new environment has differences from the old path early on but similarities later

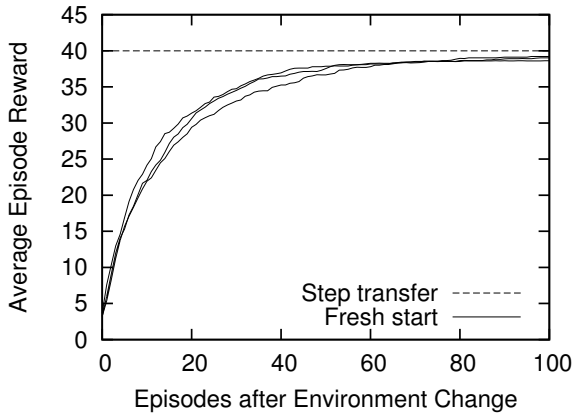

Figure 8: The *broken-action* change revisited. With the step-transfer approach, prioritized sweeping adapts with no loss in performance.

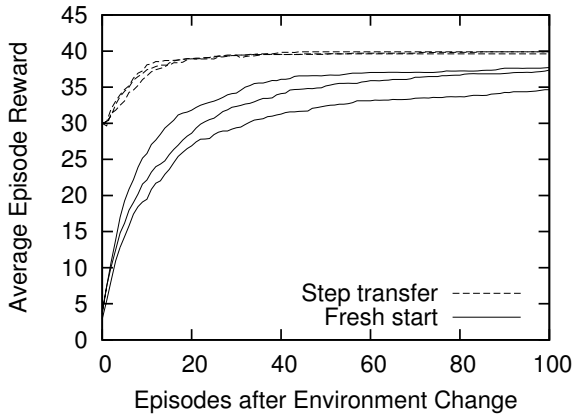

Figure 9: The *new-destination* change revisited. With the step-transfer approach, prioritized sweeping adapts effectively.

on. In this case, step transfer will not be harmful, but it may not make optimal use of the old knowledge. Another is if the optimal path in the old environment earns the same rewards in the new one, but higher new rewards become available elsewhere. In this case, any transfer algorithm might be a distraction; exploration in the new task is always important for this reason.

Conclusions

This paper pursues under-studied directions in the area of adaptation to environment change, a key aspect of transfer learning and lifelong learning. First it argues that model-based reinforcement learning is well suited to this problem, since it allows for a deeper kind of learning. Then it identifies when and why prioritized sweeping struggles with environment change: when rewards are reduced, agents unlearn expectations slowly, particularly when there are multiple paths to an expected reward. Finally it experiments with solutions that address these limitations directly in straightforward ways, proposing lightweight extensions to prioritized sweeping. These include using old knowledge selectively and quickly discarding incorrect information.

The experiments in this paper are preliminary and exploratory, providing suggestions more than conclusions. Primarily, their purpose is to indicate that methods for transfer and lifelong learning do not necessarily need to be model-free and data-driven. The Mail domain is illustrative, and lightweight transfer algorithms are likely to be beneficial in many domains that contain multiple paths to a goal state. However, broader experiments are needed for evaluation and comparison with other approaches.

The algorithms here are necessarily limited to discrete environments because they are based upon prioritized sweeping. Lightweight extensions to different model-based RL algorithms could allow for continuous state sets, and more generally, could broaden the study of this type of approach.

References

Kleiner, A.; Dietl, M.; and Nebel, B. 2002. Towards a life-long learning soccer agent. In *The 6th RoboCup Symposium*.

Lazaric, A.; Restelli, M.; and Bonarini, A. 2008. Transfer of samples in batch reinforcement learning. In *The 25th International Conference on Machine Learning*.

Minato, T., and Asada, M. 1998. Environmental change adaptation for mobile robot navigation. In *The IEEE International Conference on Intelligent Robots and Systems*.

Moore, A., and Atkeson, D. 1993. Prioritized sweeping: reinforcement learning with less data and less real time. *Machine Learning* 13.

Nowostawski, M. 2009. EVM: lifelong reinforcement and self-learning. *Computer Science and Information Technology* 12-14.

Sunmola, F., and Wyatt, J. 2006. Model transfer for Markov decision tasks via parameter matching. In *The 25th Workshop of the UK Planning and Scheduling Special Interest Group*.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *The 7th International Conference on Machine Learning*.

Szita, I., and Lorincz, A. 2008. The many faces of optimism: a unifying approach. In *The 25th International Conference on Machine Learning*.

Tanaka, F., and Yamamura, M. 1997. An approach to lifelong reinforcement learning through multiple environments. In *The 7th European Workshop on Learning Robots*.

Tanaka, F., and Yamamura, M. 2003. Multitask reinforcement learning on the distribution of MDPs. In *The IEEE Symposium on Computational Intelligence in Robotics and Automation*.

Taylor, M., and Stone, P. 2009. Transfer learning for reinforcement learning domains: a survey. *Journal of Machine Learning Research* 10.

Taylor, M.; Jong, N.; and Stone, P. 2008. Transferring instances for model-based reinforcement learning. In *The 19th European Conference on Machine Learning*.

Thrun, S., and Mitchell, T. 1995. Lifelong robot learning. *Robotics and Autonomous Systems* 15.