Exploration and Exploitation in Reinforcement Learning

Melanie Coggan
Research supervised by Prof. Doina Precup

CRA-W DMP Project at McGill University (2004)

Introduction

A common problem in reinforcement learning is finding a balance between *exploration* (attempting to discover new features about the world by a selecting sub-optimal action) and *exploitation* (using what we already know about the world to get the best results we know of). This paper will investigate the merits and disadvantages of various basic strategies for exploration and exploitation, as well as a few more sophisticated ones, all of which have been tested on reinforcement learning agents in a simple gridworld.

1. Reinforcement Learning

Reinforcement learning is a way of getting an agent to learn; for example, it may need to find the optimal path in a gridworld. The agent learns by receiving *rewards* after every action. It somehow keeps track of these rewards, and then selects actions that it believes will maximize the reward it gains, not necessarily only for the next action, but in the long run. (Sutton & Barto, Introduction).

The agent usually goes through the same environment many times in order to learn how to find the optimal actions. Balancing exploration and exploitation is particularly important here: the agent may have found a good goal on one path, but there may be an even better one on another path. Without exploration, the agent will always return to first goal, and the better goal will not be found. Or, the goal may lie behind very low reward areas, that the agent would avoid without exploration. On the other hand, if the agent explores too much, it cannot stick to a path; in fact, it is not really learning: it cannot exploit its knowledge, and so acts as though it knows nothing. Thus, it is important to find a good balance between the two, to ensure that the agent is really learning to take the optimal actions.

2. Basic Strategies

There are several types of strategies whose aim is to achieve a healthy balance between exploration and exploitation. In this section two basic types will be presented, as well as the results each give.

2.1.State-action value updating strategies (Sutton & Barto, Chapter 6)

As mentioned above, a reinforcement learning agent receives rewards as it moves through the environment. It uses these rewards for future reference; that is, when it reaches a state it already seen, it picks an action that has given it good rewards in the past.
Thus, rewards need to be stored somehow. Since several actions may be taken from each state, we store a value for each action from each state: the *state-action value*, denoted

$Q(state, action)$. This value depends in part on the reward received, in part on the current value, and can also depend on other values as well. The way of determining this value is the updating strategy. We will see two such strategies.

2.1.1.Updating strategy: Sarsa learning

Sarsa learning is an *on-policy* updating strategy. The new state-action value depends on the reward received after taking an action, on the current value of the state, *as well as* the value of the next state-action pair seen. This method was tested using the following algorithm, taken from Sutton & Barto (Chapter 6, Section 4).

```
$// s, s' \rightarrow states$
$// a, a' \rightarrow actions$
$// Q \rightarrow state\text{-}action\ value$
$// \alpha, \gamma \rightarrow learning\ parameters(learning\ rate, discount\ factor)$
1. Initialize $\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})$ arbitrarily
2. Repeat (for each episode)
    2.1. Initialize $\boldsymbol{s}$
    2.2. Choose $\boldsymbol{a}$ from $\boldsymbol{s}$ using policy derived from $\boldsymbol{Q}$ (e.g. $\epsilon$-greedy)
    2.3. Repeat (for each step of episode) until $\boldsymbol{s}$ is terminal
        2.3.1. Take action $\boldsymbol{a}$ observe reward $\boldsymbol{r}$, state $\boldsymbol{s'}$
        2.3.2. Choose $\boldsymbol{a'}$ from $\boldsymbol{s'}$ using policy derived from $\boldsymbol{Q}$ (e.g. $\epsilon$-greedy)
        2.3.3. $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) + \alpha[\boldsymbol{r} + \gamma \cdot \boldsymbol{Q}(\boldsymbol{s'},\boldsymbol{a'}) - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})]$
        2.3.4. $\boldsymbol{s} \leftarrow \boldsymbol{s'},\ \boldsymbol{a} \leftarrow \boldsymbol{a'}$
```

2.1.2.Updating strategy: Q-learning

Q-learning, unlike Sarsa learning, is an *off-policy* updating strategy. Where the new state-action value in Sarsa depends on the value of the next state-action pair *taken*, in Q-learning it depends of the *optimal* state-action pair of the next state. This method was tested using the following algorithm, taken from Sutton & Barto (Chapter 6, Section 5)

```
$// s, s' \rightarrow states$
$// a, a' \rightarrow actions$
$// Q \rightarrow state\text{-}action\ value$
$// \alpha, \gamma \rightarrow learning\ parameters(learning\ rate, discount\ factor)$
1. Initialize $\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})$ arbitrarily
2. Repeat (for each episode)
    2.1. Initialize $\boldsymbol{s}$
    2.2. Repeat (for each step of episode) until $\boldsymbol{s}$ is terminal
        2.2.1. Choose $\boldsymbol{a}$ from $\boldsymbol{s}$ using policy derived from $\boldsymbol{Q}$ (e.g. $\epsilon$-greedy)
        2.2.2. Take action $\boldsymbol{a}$ observe reward $\boldsymbol{r}$, state $\boldsymbol{s'}$
        2.2.3. $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) + \alpha[\boldsymbol{r} + \gamma \cdot \boldsymbol{max}_{\boldsymbol{a'}} \boldsymbol{Q}(\boldsymbol{s'},\boldsymbol{a'}) - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})]$
        2.2.4. $\boldsymbol{s} \leftarrow \boldsymbol{s'}$
```

2.2.Action selection strategies (Sutton & Barto, Chapter 2)

It has been mentioned already that in each state (except a terminal state) the agent must select an action. There are several ways in which to decide which action to take. The simplest of these is *greedy* selection: the agent always selects the action that the highest state-action value. This method is pure exploitation. Two more sophisticated methods, that aim to achieve a balance between exploration and exploitation, are presented in this section.

2.2.1.Є-Greedy selection

Є-Greedy is a variation on normal greedy selection. In both cases, the agent identifies the best move according to the state-action values. However, there is a small probability **Є** that, rather than take the best action, the agent will uniformly select an action from the remaining actions.

2.2.2.Boltzmann selection

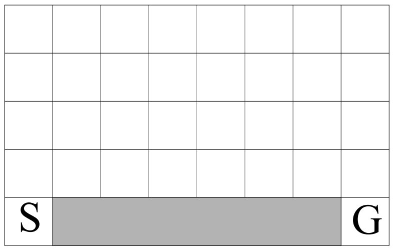
Boltzmann selection also involves probability, but takes into account the relative values of the state-action values. The probability that an action is selected depends on how it is compared to the other state-action values. So, if one value is much higher, it is most likely to be taken, but if there are two actions with high values, both are almost equally likely.
At a state ***s***, an action ***a*** is selected with probability:

$$p=\frac{e^{\frac{Q(s,a)-max_b Q(s,b)}{T}}}{\sum_a e^{\frac{Q(s,a)-max_b Q(s,b)}{T}}}$$

where **T** is called the *temperature*, and increases the exploration rate as it increases.

2.3.Results

Both algorithm shown above were tested both with **Є**-Greedy and Boltzmann action selection in order to determine which combination yields the most promising results. They were all tested in a simple gridworld similar to that described in Sutton & Barto (Chapter 6, Section 5, Figure 6.12).



The agent starts at point S, and can move up, down, left or right. For each action taken, the agent gets a reward of -1. The goal G gives +100, and the gray zone is a 'cliff', which gives a reward of -100 and sends the agent back to the start. The idea is, of course, for the agent to learn to reach the goal in the least number of steps. The results obtained were averaged from 100 trials of 500 episodes each.

Whether using Sarsa learning or Q-learning, **Є**-Greedy was pretty much the same. As the value of **Є** gets smaller, the higher the average reward gets. Technically, a lower value for **Є** should be slower, but the difference does not show in the small, 5x8 world.

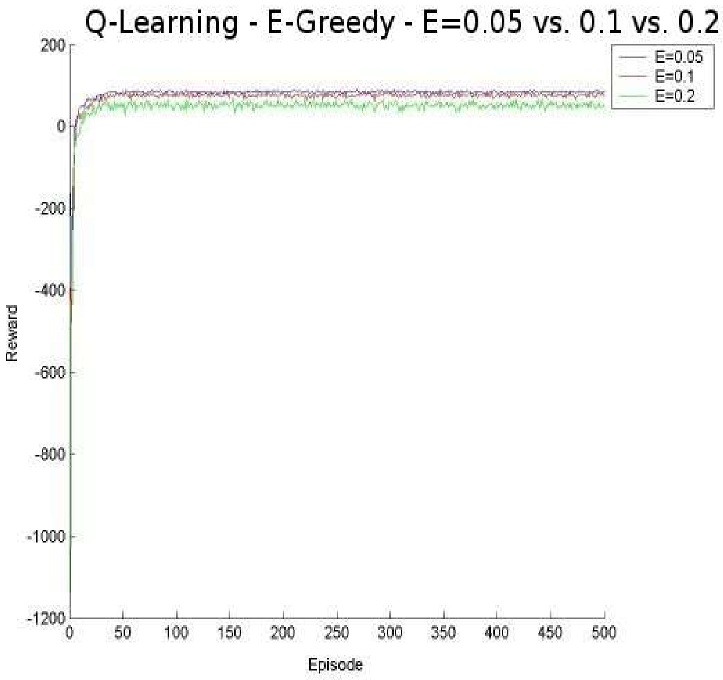


Figure 2.1

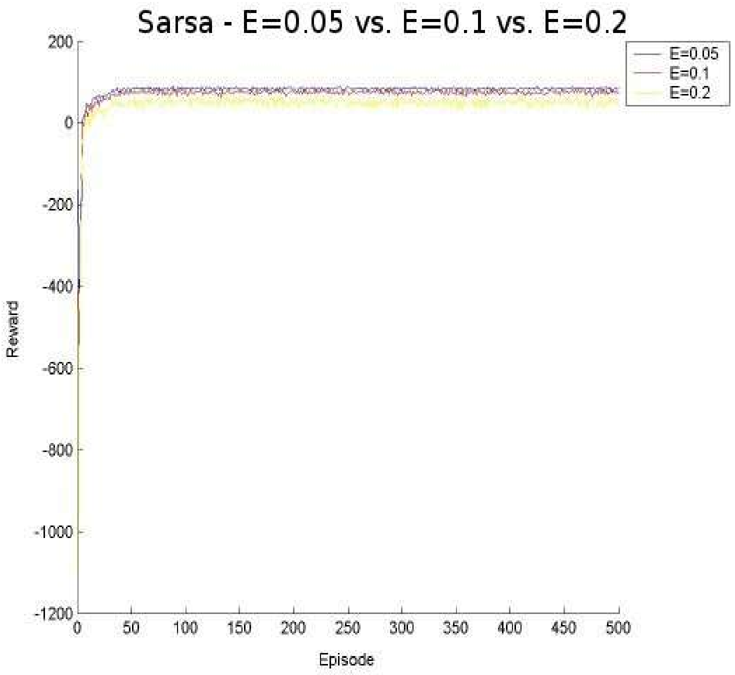


Figure 2.2

Boltzmann learning shows rather different results. For a low temperature, there is not much difference, except that Sarsa learning is somewhat slower and more stable (see Figure 2.3). However, as the temperature gets greater, so does the difference. Q-learning gets a higher peak early on, but quickly falls, whereas Sarsa never reaches as high, but stays relatively stable, and gives better results in the end (see Figure 2.4). In both cases, a lower temperature gives better results: better stability and higher reward.
Between **Є**-Greedy and Boltzmann selection, it seems clear that Boltzmann is the better of the two. We can comparing **Є** = 0.05 and **T**=10, both of which gave the best results. While Boltzmann does learn slower in the beginning, it eventually gets better; not only this, but it is almost perfectly stable (see Figure 2.5). An agent using **Є**-Greedy can never achieve this, since it keeps exploring regardless of how many times it received very low rewards.

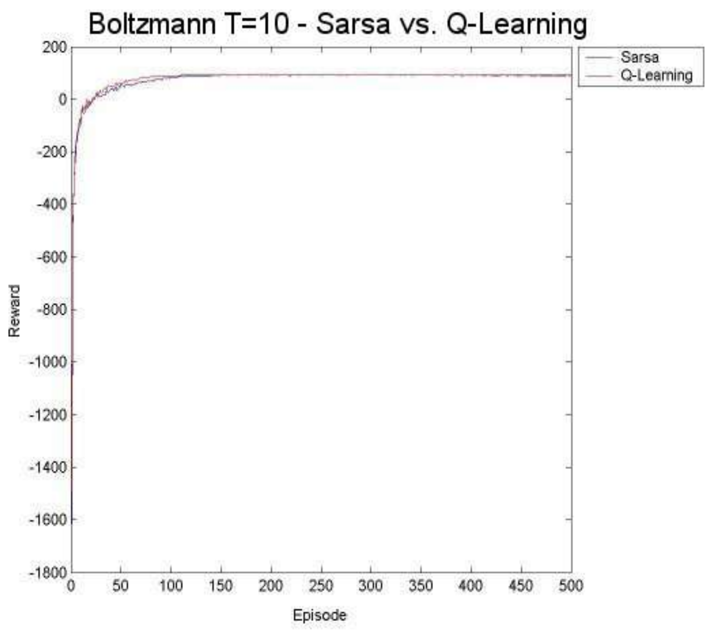


Figure 2.3

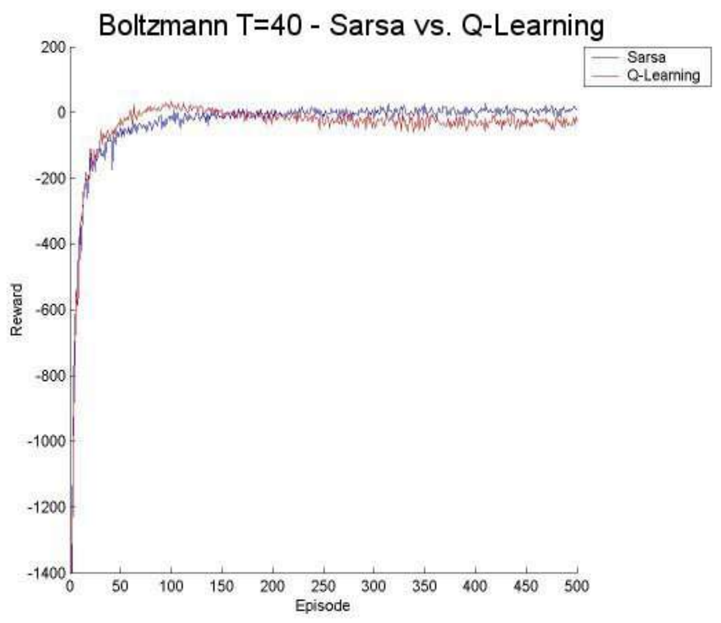


Figure 2.4

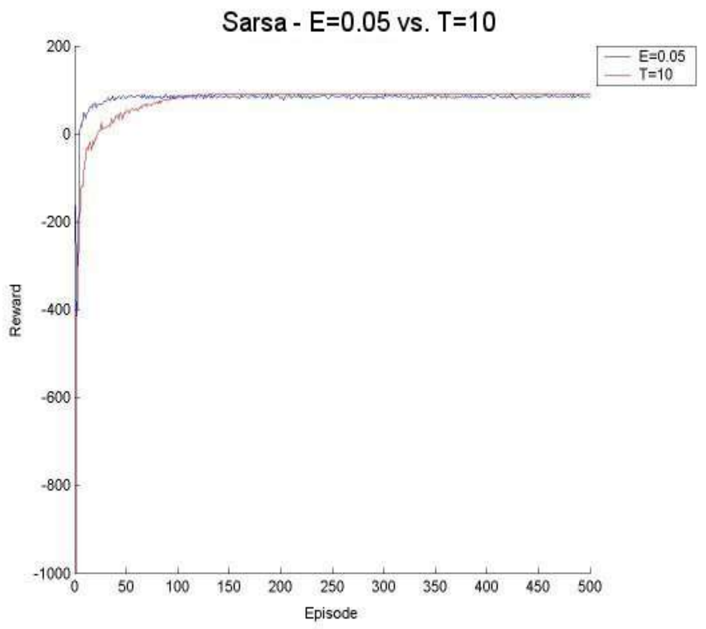


Figure 2.5

Other than the reward, there is also the question of which path the agent takes. An agent with Sarsa learning will learn to use a different path depending on the exploration rate: for a low temperature, it will stay close to the cliff, and moves further away as the temperature increases. An agent using Q-learning will always select the optimal path (that is, the path closest to the cliff).

This brings us to some conclusions, keeping in mind that we want a *sophisticated* agent, one that will not simply avoid cliff most of the time, but that will avoid them all of the time.
First of all, if, we cannot use **Є**-Greedy. Boltzmann selection, even though it is slower, is much better suited to the task: if a move is very bad compared to others, the probability of selecting it are correspondingly low. **Є**-Greedy never varies the probability of choosing moves, so it will inevitably return, by random chance, to spots that have been repeatedly bad.
Secondly, Sarsa learning has more desirable properties than Q-learning. For one, an agent using Sarsa learning performs better when the temperature increases. But, even more important, it learns to avoid dangerous areas. In the case with **T** = 40, the agent chose safer and safer paths as episodes went on. The agent with Q-learning though, always tried to stick to the shortest path, and a result went into the cliff more often, getting poor results.
For these reasons, the experiments in the following sections are all made with agents using Sarsa learning and Boltzmann action selection.

3. Advanced Techniques

A few techniques whose aim is to improve the agent's performance will be introduced in this section.

3.1.Increased learning rate in disaster situations

As mentioned in Section 2.1, reinforcement learning agents store state-action values that are updated at each time-step of an episode. For Sarsa learning, the update is rule is shown in Section 2.1.1, on line 2.3.3 of the algorithm:

```
2.3.3.    $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) + \alpha[\boldsymbol{r} + \gamma \cdot \boldsymbol{Q}(\boldsymbol{s}',\boldsymbol{a}') - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})]$
```

The parameter **α** is called the *learning rate*. This determines how much the new state-action value tends towards the newly obtained reward and value of the next state-action pair. The greater **α**, the more the state-action value tends towards new information. As a general rule, higher values of α learn faster, but end up receiving slightly lower rewards.

The increased learning rate technique consists in having *two* learning rates. One for normal situations and rewards, and another, greater one for disaster situations. So, when an agent moves into a cliff (or other disaster situation), the new state-action value tends much more to the value of the received reward. We can replace the algorithm step shown above by:

```
2.3.3.    if (disaster situation)
2.3.4.        $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) + \alpha_{\mathbf{fast}}[\boldsymbol{r} + \gamma \cdot \boldsymbol{Q}(\boldsymbol{s}',\boldsymbol{a}') - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})]$
2.3.5.    else
2.3.6.        $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a}) + \alpha[\boldsymbol{r} + \gamma \cdot \boldsymbol{Q}(\boldsymbol{s}',\boldsymbol{a}') - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})]$
```

The idea behind this technique is that the agent should get the benefit of a high learning rate, while avoiding disadvantages. It should quickly learn to avoid very bad areas while still exploring the rest of the environment.
This technique can also be applied to goal situations. It can also be advantageous to have the agent learn the goal's position more quickly. However, if there are additional goals, this may prevent the agent from finding them.

3.2.Eligibility traces (Sutton & Barto, Chapter 7)

As shown in the algorithms above, at each step only one state-action value is updated, the last state-action pair seen. With eligibility traces, the values of the sequence of state-action pairs that led to the current state are updated. For each state-action pair, a new value, the *eligibility trace* is stored. This value increases whenever the state-action pair is seen, and decrements slowly at every other time step. The eligibility trace determines to what extend the state-action value is updated: states with high eligibility traces tend more towards the new information, whereas states with lower eligibility traces hardly change at all.
The following is the algorithm used in testing, taken from Sutton & Barto (Chapter 7, Section 5). It replaces line 2.3.3. in the Sarsa algorithm above.

```
for all $s, a$ (at time step $t$ )
    $Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha \cdot \delta_t \cdot e_t(a,s)$
where
    $\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$
    $e_t(s,a) = \begin{cases} \gamma \lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma \lambda e_{t-1}(s,a) & \text{otherwise} \end{cases}$
```

λ is the parameter of the eligibility trace. The greater it is, the longer the sequence of values of state-action pairs updated.

The purpose of eligibility traces is to propagate rewards to the state-action values faster. Without the traces, only the state-action pair right before the goal or cliff is updated. With, most of the path leading to it is updated. Hence, the next time the agent arrives on a state in that path, it is more likely to avoid paths leading to the cliff, and to take paths that led to a goal.
Ideally, this will improve the performance of the agent in the long run (in the beginning, it may cause to avoid paths that lead to the goal if it veered off track and got a low reward on the way)

4. Experiments

Experiments were performed to test the above techniques, as well as test a few other features of reinforcement learning, and of the environments. The environments tested and the experiments themselves, as well as the results will be presented in this section.

4.1.Environments

With the exception of the ones mentioned above in the “Basic Strategies” section, all experiments were performed on one of two environments. Both are gridworlds, one small and one large. Their sizes were 24×15 (Figure 4.1) and 30×30 (Figure 4.2), respectively. Their configurations are as follows:

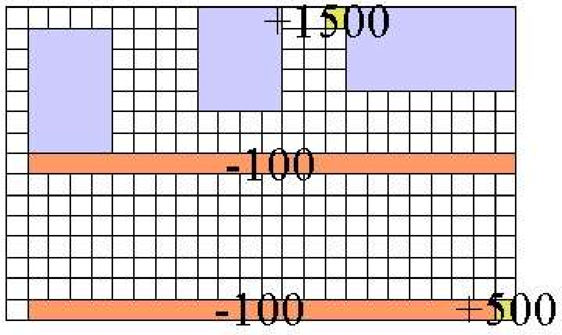


Figure 4.1

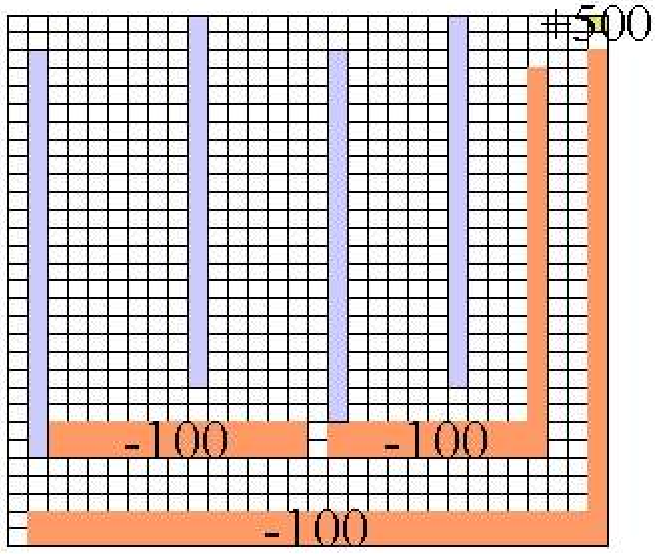


Figure 4.2

In both cases cliffs are represented in orange, with negative values (representing the negative reward). Obstacles, areas that the agent cannot pass or go on, are in light blue.

Each world is designed to test different things. The small goal has a nearby, fairly easy to reach goal, and another bigger goal further away. The idea is to see which *goal* the agent will find, and how temperature, increased disaster-learning rate, and eligibility trace influence this.
The large world has only one goal, with two paths leading to it. The shorter path is closely lines by cliffs, a dangerous route. The agent can also use the safe path, but

because of the obstacles it is much longer. In this case, the goal of the experiments is to see which *path* the agent will take.

4.2.Experiments and results

The experiments consisted of running an agent through the environments multiple times, while slowly changing a single parameter, to see its effect.
The effect of the increased learning rate was tested in the small gridworld; temperature, and eligibility traces in both environments. In addition, the effect of different rewards for the goal was also tested, since the value originally set (+500) was too low, and regardless of all other parameters the agents would not learn.

4.2.1.Experiment 1: increased learning rate

The first experiment tests the value of the increased learning rate in disaster situations. In this case the environment is the small gridworld, as shown in Figure 4.1. The agent uses Boltzmann action selection with a temperature **T**=20, Sarsa learning with $\boldsymbol{\alpha} = 0.1$, and eligibility traces with $\boldsymbol{\lambda} = 0.3$. $\boldsymbol{\alpha}_{\mathbf{fast}}$ was increased 0.1 by 0.1, starting from 0.2.
Generally, the results are as predicted. The agent quickly learns not to go into cliff areas; after going only a few times on any cliff square, it is not likely to ever go there again, a clear advantage in the static gridworld. As $\boldsymbol{\alpha}_{\mathbf{fast}}$ is increased, the agent needs to go fewer times in the cliff before learning not to return.
The best value for $\boldsymbol{\alpha}_{\mathbf{fast}}$ seems to be around 0.5. Each 0.1 increase before 0.5 makes a large difference in the learning rate, but after this value the difference is smaller and smaller. It seems, then, that 0.5 is a good value to take; this environment is static, but in a non-static environment a high value for $\boldsymbol{\alpha}_{\mathbf{fast}}$ might stop the agent from ever returning to a square that is bad only a few times.

This technique results in high rewards much faster. However, it does not increase the maximum reward gained. In the end, the agent performs as well, but peaks faster (see Figure 4.3).

4.2.2.Experiment 2: temperature

Next comes a series of experiments that examine the effect of the temperature parameter. For these experiments, the agents were made to go through the environments several times, with varying temperature. All other parameters remained the same, for comparison.

The first series looks at the difference made by temperature variations in the small environment. The agent uses Sarsa learning with $\boldsymbol{\alpha} = 0.1$, $\boldsymbol{\alpha}_{\mathbf{fast}} = 0.5$, no eligibility traces, and temperatures varying from **T**=1 to **T**=30. At first glance, it seems that lower temperatures perform better: they are faster to learn, and receive higher and more stable rewards once they settle on an 'optimal' path (see Figures 4.4 and 4.5).

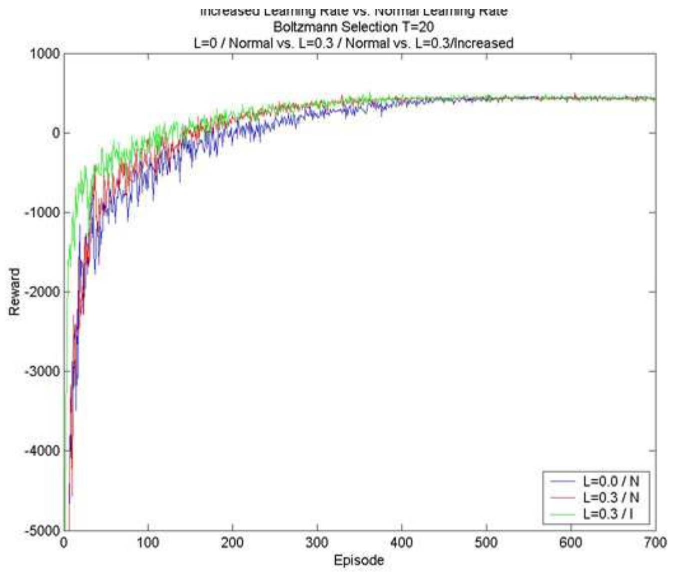


Figure 4.3: Note the difference between the green (increased learning rate) and red (normal learning rate) curves, particularly in the first few trials.

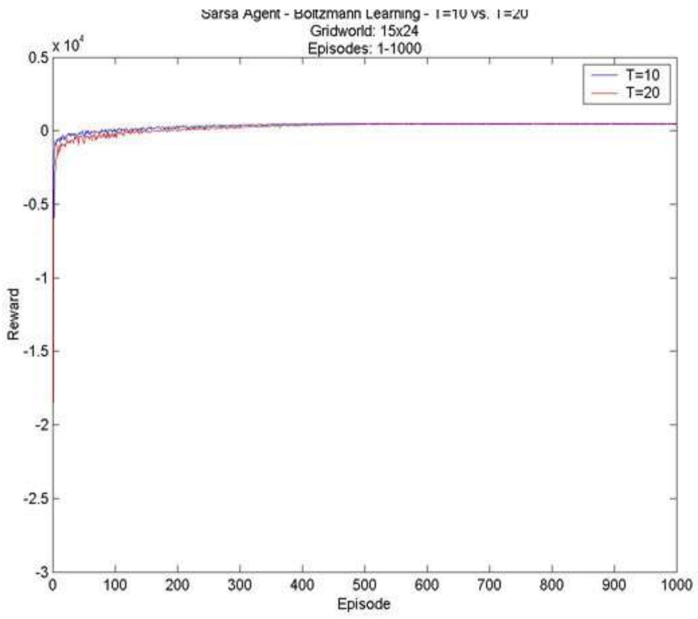


Figure 4.4

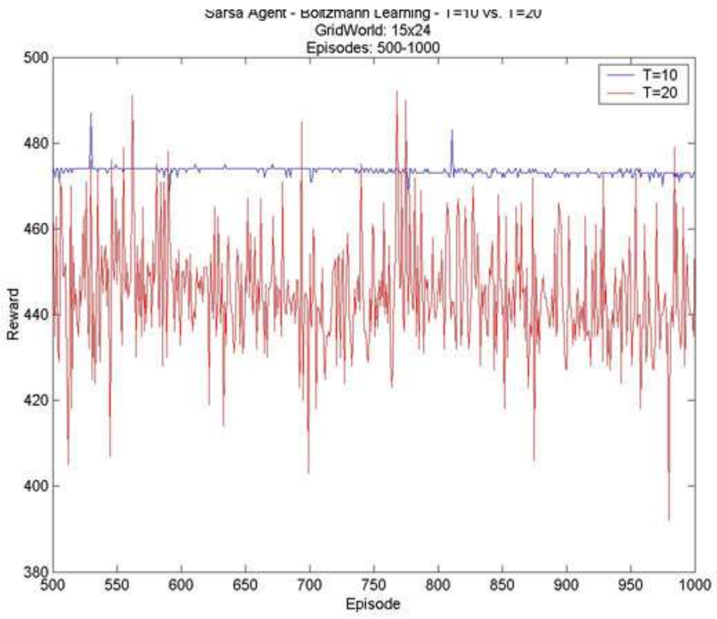


Figure 4.5

However, in the environment all these low-temperature agents find the nearer, lower reward. Higher temperature agents, that explore more, also find the lower reward, at first. But an agent with a high enough temperature (in this case, **T**=30), even if it has already found an 'optimal' path and its received rewards have stabilized, will eventually find the better goal, and start using that path more and more often. The agent with **T**=30 finds the better goal after about 8000 episodes, after seemingly have peaked at 400 episodes, and in

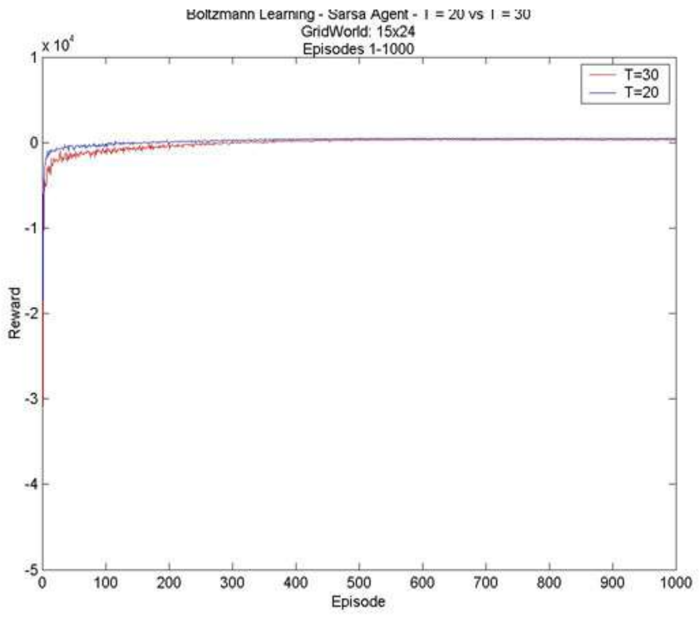


Figure 4.6

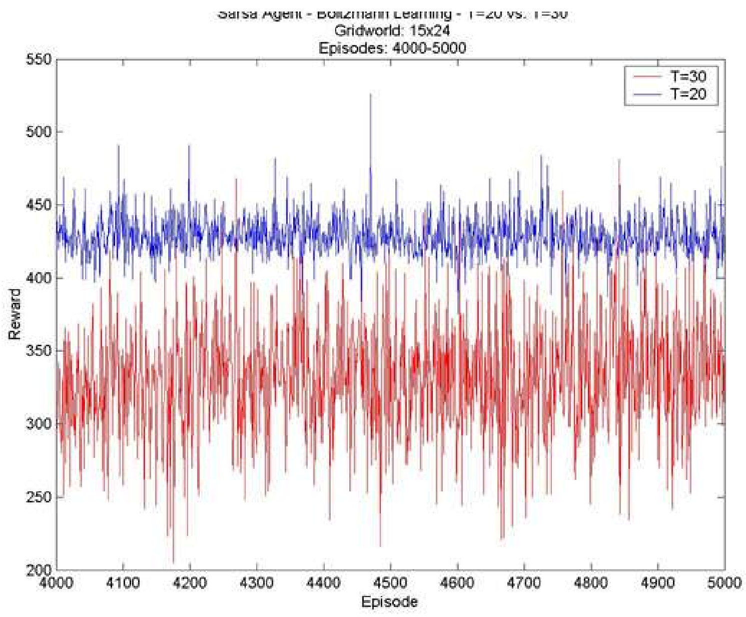


Figure 4.7

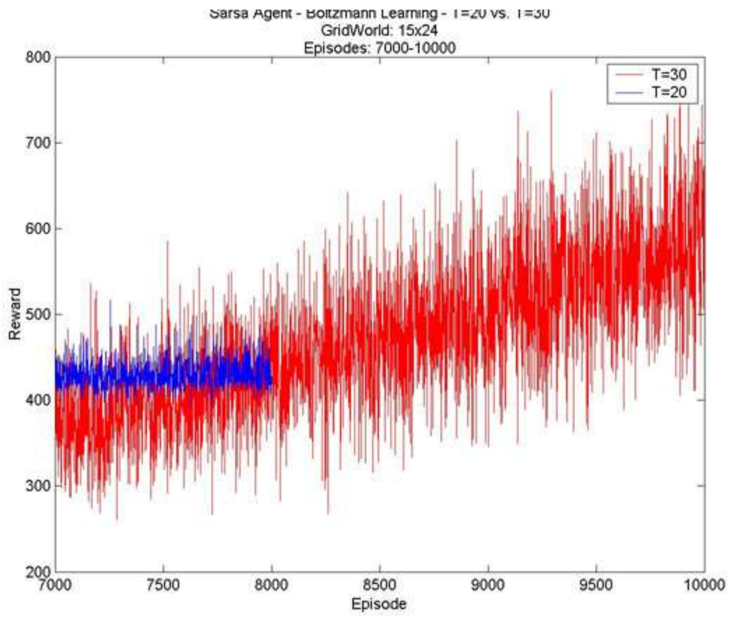
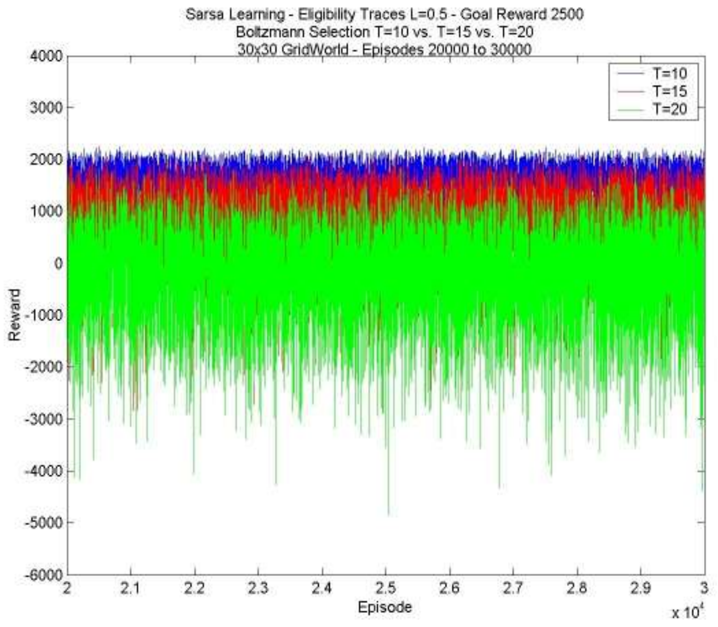


Figure 4.8

the end earns much better rewards than **T**=20 (see Figures 4.6, 4.7, 4.8).

In the large environment, with no 'secret' goal to find, the results were very clear. For an agent using Sarsa learning with **α** = 0.1, $\boldsymbol{\alpha}_{\mathbf{fast}}$ = 0.5, eligibility traces with **λ** = 0.5, temperatures varying from **T**=10 to **T**=20, and a goal reward of +2500 (as opposed to 500 in Figure 4.2), lower temperatures were better in all respects. As in the small environment, agent using smaller temperatures learn faster and stabilize at higher values, and vary much less (see Figures 4.9 and 4.10). However, there is no hidden goal, so high



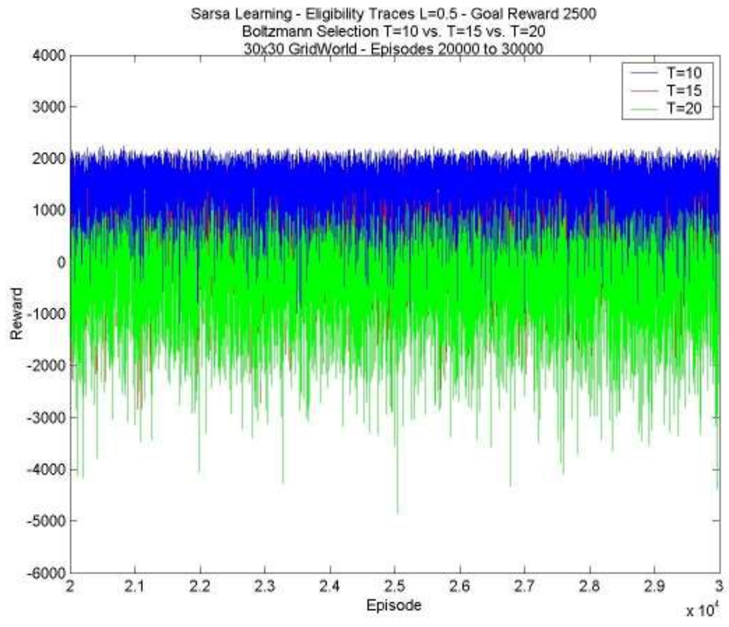


Figure 4.9: Note how the blue curve (T=10) is much higher than the green curve (T=20), and has much smaller variance

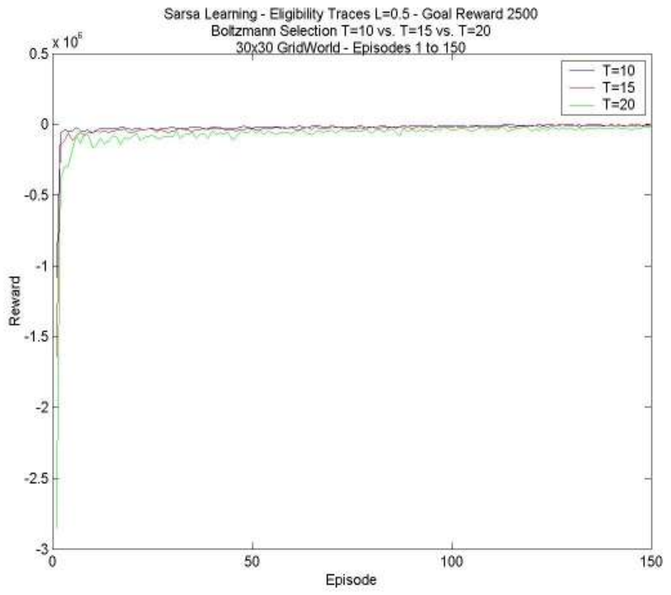


Figure 4.10

temperatures do not eventually lead to an increase.

The results found on both world lead to somewhat problematic conclusions. *If* there is more than one goal, it is clearly advantageous to use higher temperatures; this will help the agent find all the different goals, and settle on a path that is best (most probably straying every so often). On the other hand, on a world with a single goal, or perhaps even

a world with multiple goals, but with the nearest being the best, low temperatures are much better.
It seems that there is no way to set a good temperature right from the start. It is necessary to use fore-knowledge about the environment (if any) to decide what is best, or run an agent with a high temperature at first to get an idea of the environment.

4.2.3.Experiment 3: eligibility traces

This series of experiments deal with eligibility traces. As in the other experiments, agents were made to go through an environment with all values remaining the same, except for the parameter **λ**. As for the experiments with temperature, both the small and large environments are used.

The experiments on the small environment were conducted with an agent using Sarsa learning with **α** = 0.1, **α_{fast}** = 0.5, temperatures varying from **T**=20 to **T**=30, eligibility traces with **λ** varying from 0.1 to 0.4. As was mentioned in the preceding section, low temperatures could not find the second goal in this environment. According to the test results, this is precisely what eligibility traces help with. Without eligibility traces, an agent with temperature 20 did not find the second goal. *With* eligibility traces, it can. With **λ** = 0.3, the agent finds the second goal after about 7500 episodes (see Figure 4.11). Also, the traces help with the speed at which the agent finds the second goal. With **λ** = 0.4, the agent with **T**=20 finds the goal after only about 6000 episodes; for each increase of **λ** the agent with **T**=30 finds the second goal a little bit faster; however, the difference between each increase becomes smaller and smaller (see Figure 4.12)

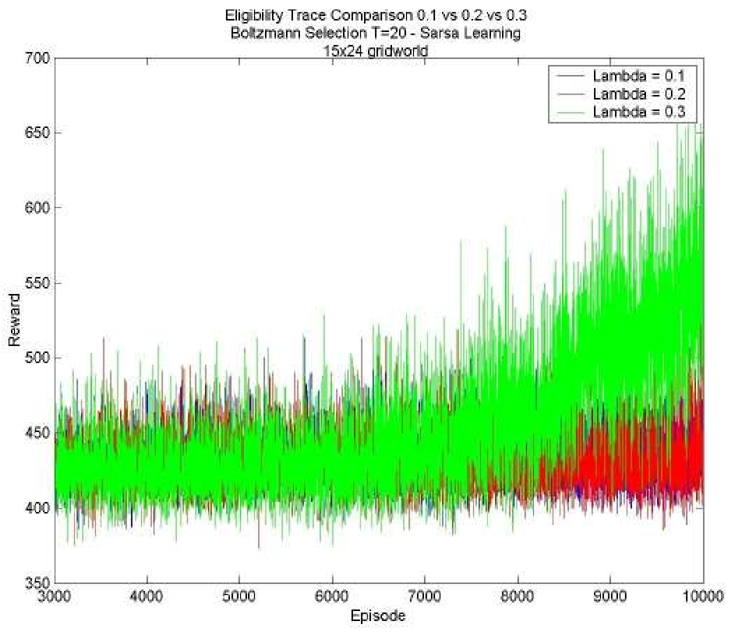


Figure 4.11

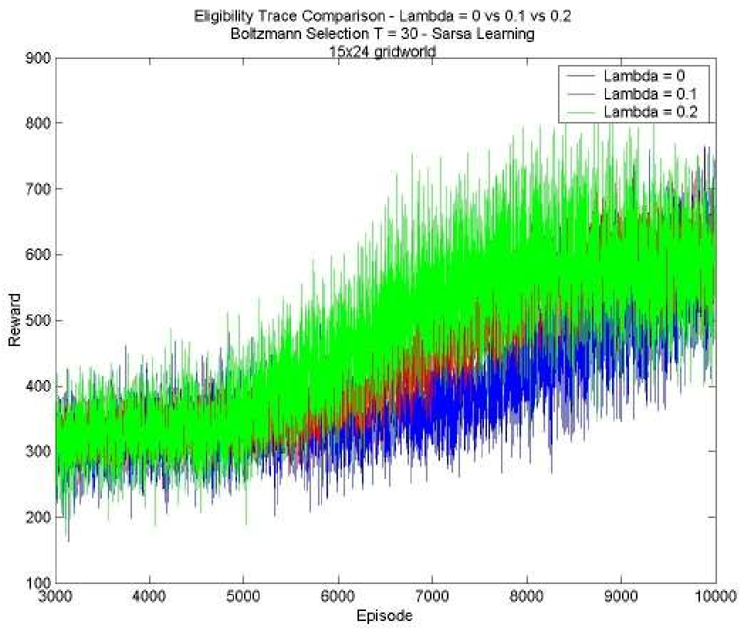
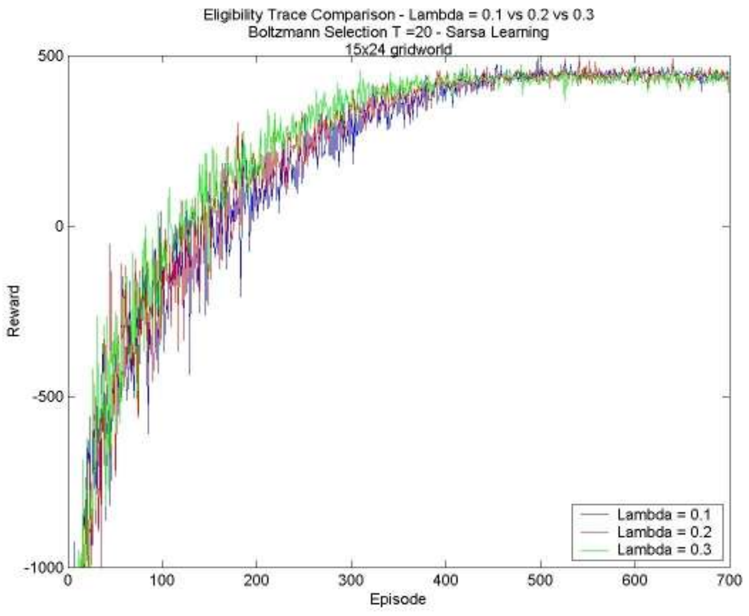


Figure 4.12

The tests also show that the traces can help improve the speed at which the agent learns. In both cases (**T**=20 and **T**=30), the agents with higher **λ** values saw better results in the first few trials (see Figures 4.13 and 4.14). They do not, however, have any impact on how *well* an agent finds its preferred goal. Before and after the increase in rewards due to the agent finding the second goal, the rewards are all the same: only the speed at which the agent gets to the better rewards improves.



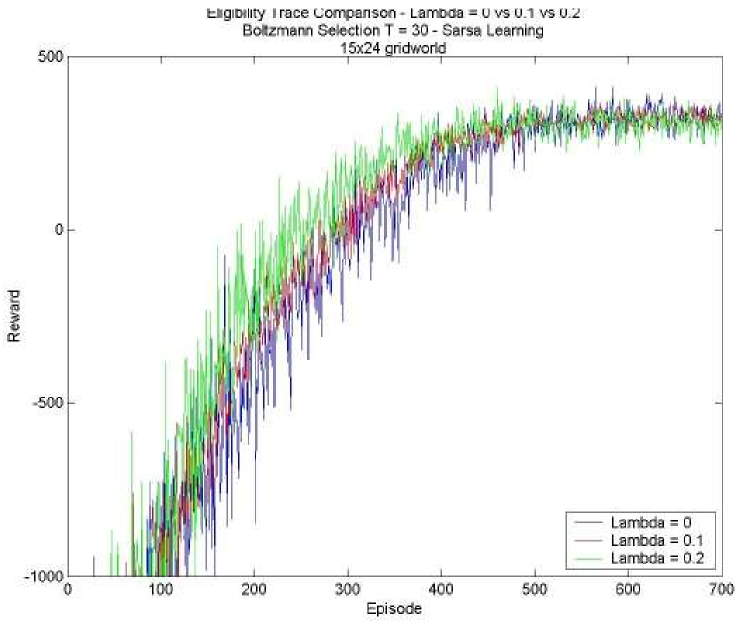
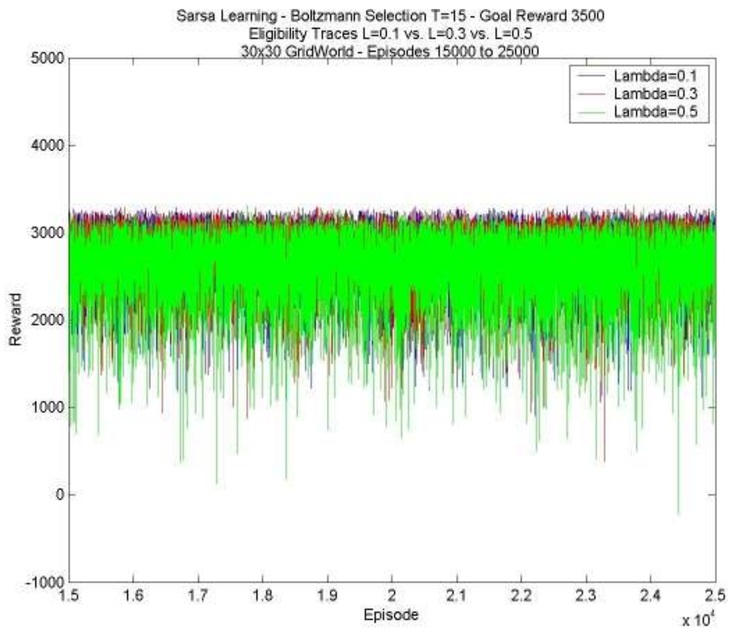


Figure 4.13

Figure 4.14

The experiments in the large environment are all performed using an agent using Sarsa learning with **α** = 0.1, $\boldsymbol{\alpha}_{\textbf{fast}}$ = 0.5, a temperature of 15, a goal reward of +3500, and eligibility traces with **λ** varying from 0.1 to 0.6.
In these cases, the value of **λ** changes almost nothing. When the agents stabilize, all of them are collecting rewards of the same value (with higher values of **λ** showing a *bit* more variance) (see Figure 4.15). There is a small difference as well in the initial learning rate, higher values of **λ** learning a bit slower, but not very noticeably (see Figure 4.16). One important thing must be noted, however: the difference between **λ** = 0.5 and **λ** = 0.6 is enormous. The latter fails completely, taking weeks to finish a single trial of 70000 episodes (whereas the agent with **λ** = 0.5 completed this in less than a day), and receiving extremely poor rewards: the agent is confused and the long trace causes it to travel almost circularly.

One final thing to note on eligibility traces: the added computation time is not negligible. The higher the value of **λ**, the longer it takes to compute. It can be useful to set a limit for updating state-action values. If the eligibility trace is smaller than the limit, you do not update it. This can be important, since low values really don't change much, and in large



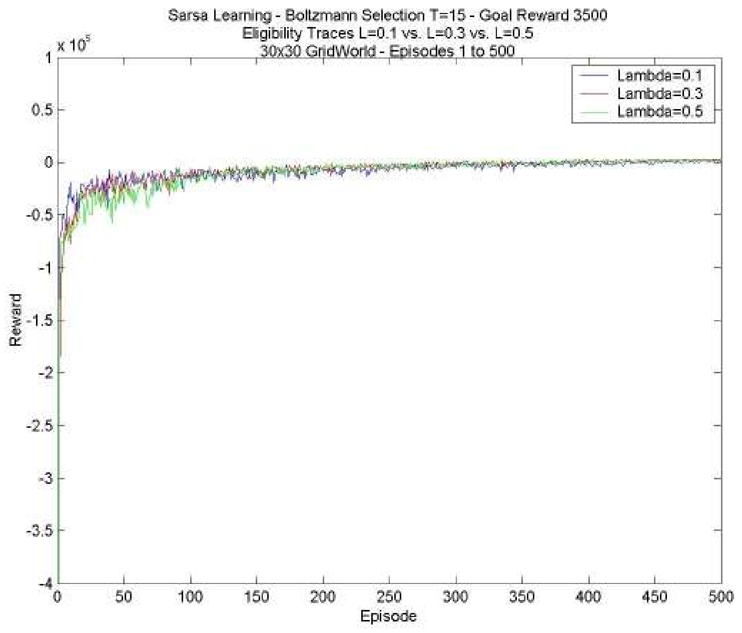


Figure 4.15 **Figure 4.16**

environments the trace can get very long.

Some useful conclusions can be drawn from these experiments, particularly when taken together with experiment 2, dealing with temperatures. As was seen in the small environment, eligibility traces can help a lower temperature find its way to multiple goals. So, using the traces allows us to set the initial temperature *lower*, which was very advantageous in the large environment.
Another point to note is that the value of **λ** must be kept low enough, around 0.3; otherwise the agent may end up learning nothing at all

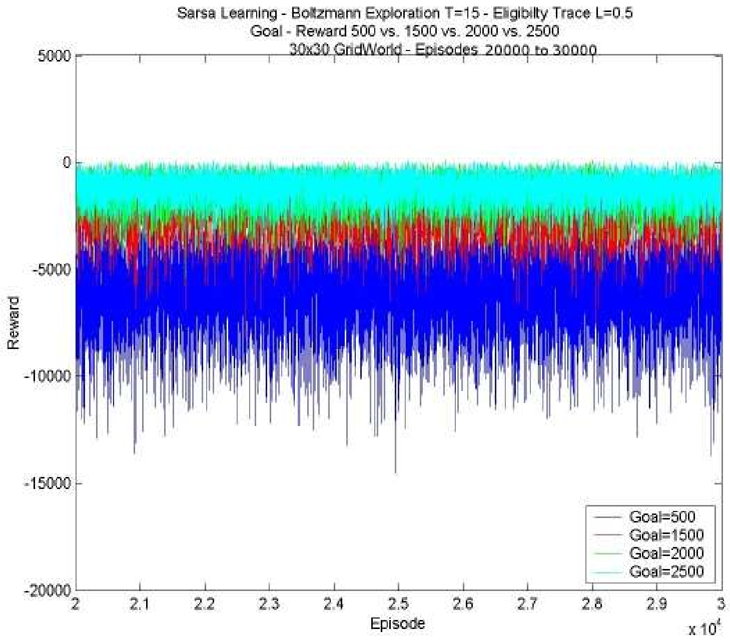
The best way to use of eligibility traces seems to be in conjunction with low temperatures. They allow the high, stable rewards, while still finding new goals, if any. The increased computation time can be well worth the results, particularly when a limit is set on how high a eligibility trace must be to update a value.

4.2.4.Experiment 4: goal rewards

This is the only experiment that was not actually planned before hand. But, none of the agents were able to learn anything on the large environment at first (as shown in Figure 4.2). Not only this, but changing values for **λ** or **T** did not change anything. It turns out that the value of the goal reward is critical. No matter what the other parameters are, the Sarsa cannot learn in an environment where it does not receive enough positive feedback.

The experiments are performed using an agent using Sarsa learning with **α** = 0.1, $\boldsymbol{\alpha}_{\textbf{fast}}$ =

0.5, a temperature of 15, eligibility traces with **λ** =0.5, and goal rewards varying from +500 to +3500.
The results of the experiment show that the value of the goal reward makes a big difference. The agent collects much greater rewards. For an increase of only 1000, the agent was able to go from earning rewards of -5000 to earning positive rewards, a difference much greater than the extra 1000 given by the goal. As the goal reward increases, the agent performs better and better, up to a certain point (see Figure 4.17). The goal reward cannot be increased indefinitely for infinitely good performance. Eventually, the agent's performance peaks; however, even though it is no longer earning better rewards, it has a tendency to earn more *stable* rewards (see Figure 4.18).



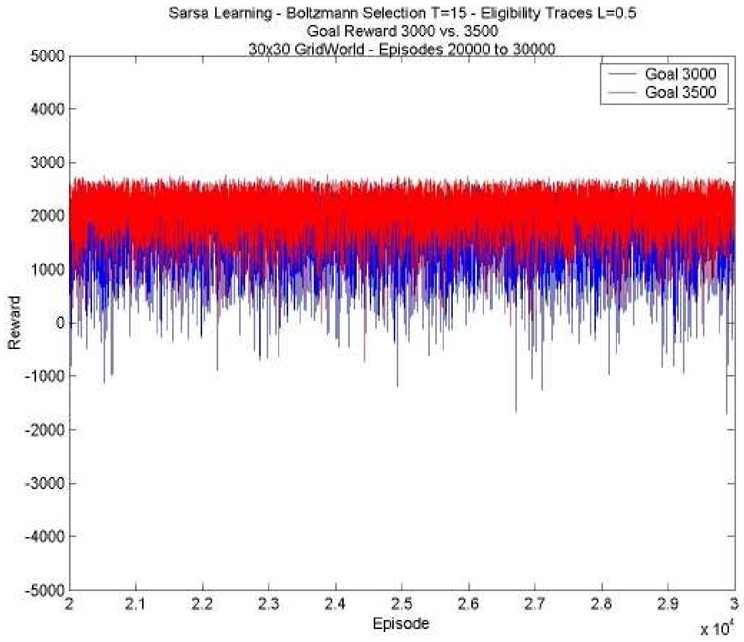


Figure 4.17 **Figure 4.18**

A certain amount was subtracted from each curve, to make up for the difference between the goal rewards. All were brought down to a reward of 500 for Figure 4.17 (i.e. 1500 was subtracted from the 2000 curve) and to 3000 for Figure 4.18

It becomes clear that setting an appropriate goal reward is quite important. We already know what happens if it is too low, but it can also be too high. The stabilizing effect noticed when increasing the reward is due to the state-action values leading to that goal increasing faster when the reward is larger. If the state-action values are much greater along that path, the probability of taking another path becomes smaller and smaller, which means that the agent will not explore as much. As we saw in the small environment, exploration is critical in certain environments.

These experiments show the importance of setting rewards that are neither too big nor too small. In these cases the rewards were hard coded into the environment. However, in cases that allow the agent's programmer to decide on a reward (for example, what reward to give when a robot bring you coffee in the morning), it might be useful to test a range of rewards and select one of the smallest values that gives high performance (if the agent needs to explore).

5. Conclusions

The techniques tested all showed some kind of promise: improving performance, increasing the learning rate, helping to find better goals. All this is very useful for a reinforcement learning agent.
However, these techniques require a certain amount of fine-tuning of the parameters involved, and in some environments it is even preferable *not* to use some (in particular, eligibility traces in a one-goal environment). This somewhat limits the usefulness of the techniques, in that they cause the reinforcement learning agent to require more supervision.
In general, the various techniques can bring significant improvement, when a little supervision is possible, or if there is prior knowledge about the environment, and when used with appropriate parameters.