Markov Decision Processes

Gary Saavedra

gsaavedra3@gatech.edu

Introduction

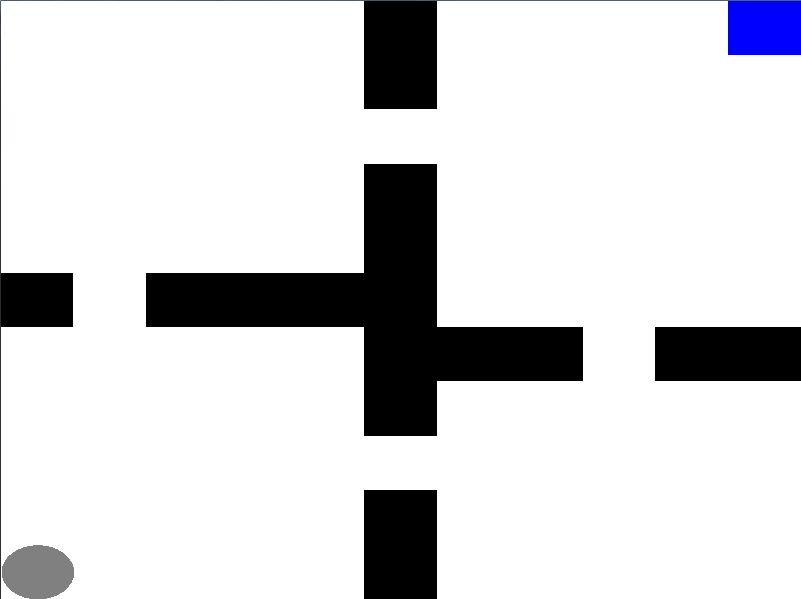
When attempting to learn a policy for an MDP there are two types of algorithms that can be deployed, a planner and a learner. A planner can be used when there is a model for the MDP the agent is trying to move through. Specifically there must be a transition function and a reward function. A learner can learn a policy even when there is no model. It does so by exploring the space. For this assignment BURLAP was used to implement Value Iteration (VI), Policy Iteration (PI), and SARSA. Each algorithm was run over 10 trials to average out some of the variance. For the choices of MDPs, a small grid and a large grid world were used.

Small Grid World

A small 10 x 10 grid world with four rooms was used for the small MDP. It is shown in Figure 1. The small grid world has a total of 104 states. The agent, the grey dot in the figure, starts in the bottom left corner and works its way to the blue goal state in the top right corner. The transition function is as follows: there is an 80% chance the agent will go in the direction intended and a 6.7% chance of moving in a direction that is not intended. The reward function gives a reward of 100 when the agent reaches the absorbing goal state, ending the game. Otherwise, the agent receives a reward of -1 for entering any other state. This reward structure gives the agent motivation to end the game as quickly as possible.

*Why is it interesting?*

The small grid world will show the strengths and weaknesses of the planning algorithms. Specifically it will show the faster running times of PI and how the VI and PI can effectively optimize a policy in small space in a short amount of time. In addition, it will point out some of the shortcomings of the SARSA algorithm. It allows for an analysis over many trials because there is not as much time needed. It will also provide a basis with which to compare the large MDP with.



*Figure 1: Small Grid World*

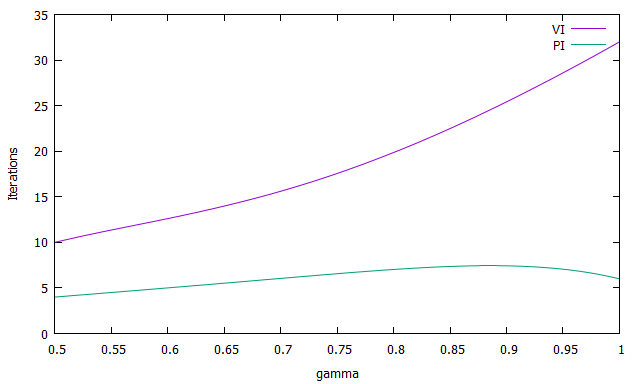
Value Iteration

VI is a planning algorithm that utilizes a model of the MDP. Specifically, it takes into account the transitional probability and reward functions to search the space and maximize reward. It uses Bellman’s equation to iteratively update the utility of a state based on the utilities of the neighboring states. Bellman’s equation is recursive in that every utility update for a state requires the utility of neighboring states which in turn require the utilities for their neighboring states. The amount of recursion through successive neighboring states is determined by the discount factor. For VI in BURLAP, a greedy Q policy is used which plans from the input state and returns a policy. This policy greedily selects actions with the highest Q value and breaks ties uniformly randomly.

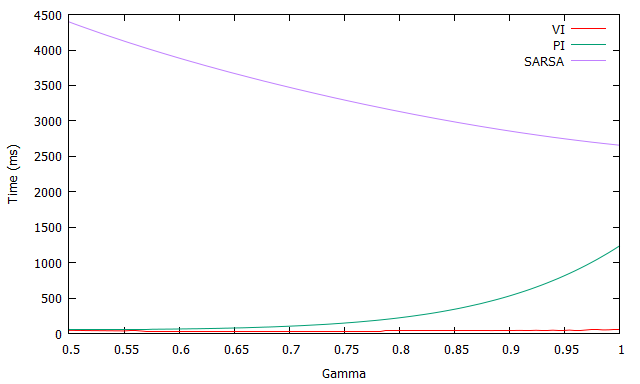
Figure 2 shows an iterations vs gamma graph for both PI and VI. Figure 2 represents the number of iterations needed for convergence for a specified gamma. Gamma throughout this paper represents the discount factor. Figure 2 shows the VI always has a larger number of iterations no matter the choice of gamma. The number of iterations required for VI grows exponentially with increasing gamma. This is because the VI is required to look farther into the future to estimate utilities. Decreasing the discount factor decreases the number of iterations to find the optimal policy. But this is because the recursive application of the discount factor is making the difference between the later value functions smaller than delta much earlier than it would with a large discount value. This leads to a weak policy being found which in turn increases the number of iterations it takes for the agent to reach the absorbing state. This is because the optimal policy will only get an optimal value for the first few squares. After that the agent must choose randomly amongst the remaining squares. This can especially be seen with discount=0.99.

Figure 3 shows a time vs gamma graph for all three algorithms. Figure 3 represents the amount of time needed for the algorithm to converge for a specified gamma. VI outperforms PI for all gamma choices greater than 0.6. PI only outperforms VI for 0.5 < gamma < 0.6. The time needed for convergence for VI increases as gamma increases. At gamma = 0.6 the time needed is 24.8 ms. The time increases as it moves up to gamma = 0.99. At gamma = 0.99 the time needed is approximately 52 ms. The process of recursively updating all the utilities is responsible for most of the time needed for VI. VI is O(A\*S2) in time complexity for each iteration where A is the total number actions from a state and S is the total number of states. There is one exception to an increasing time with increasing gamma. At gamma = 0.5 the time needed is 44.4 ms. The large discount of future reward may cause the agent to wander more and thus take a longer time to converge. The large discount of future reward may hinder the propagation of positive utilities from the goal states. Thus, VI needs more time to improve the policy. Beginning at gamma = 0.6 the agent is considering future reward enough that it tends to not wander.

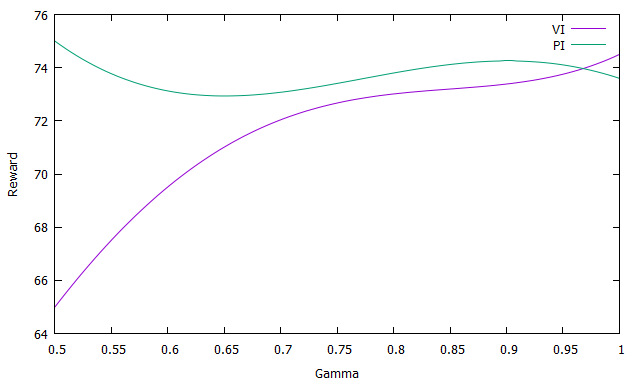
Figure 4 shows a plot of rewards vs gamma for VI. The highest reward of approximately 75 is achieved when gamma = 0.99. In an ideal situation if VI is able to get to the goal without unintentionally moving in a wrong direction it can get there in 21 steps which would result in a reward of 79. VI comes very close to converging to an optimal answer beginning at gamma = 0.7. Given the stochastic nature of the grid the VI has likely converged on the optimal policy resulting in a reward of 75. The loss of four points from the max reward is due to a few steps in the wrong direction as determined by the transition function. Comparing Figures 3 and 4 we can see that although VI is finding a policy in less time with a smaller gamma it is a weak policy. At gamma = 0.6 the average reward is only 69. Because the agent is not looking at enough successive future states to estimate utilities, the states near the starting area are not updated optimally as shown in Figure 6. A few of the states in the starting area have a policy arrow pointing in non-optimal directions. These non-optimal actions cause the agent to wander a bit before reaching a state where the optimal policy has flowed outward from the goal. While wandering the agent accumulates a few extra -1 rewards resulting in a lower final reward. However, if we wished to reduce time we could use a smaller gamma and still achieve near-optimal rewards. For example, we could use a gamma of 0.7 and converge to a nearly optimal answer while saving time.



*Figure 2: Iterations vs gamma for PI and VI, Small grid*

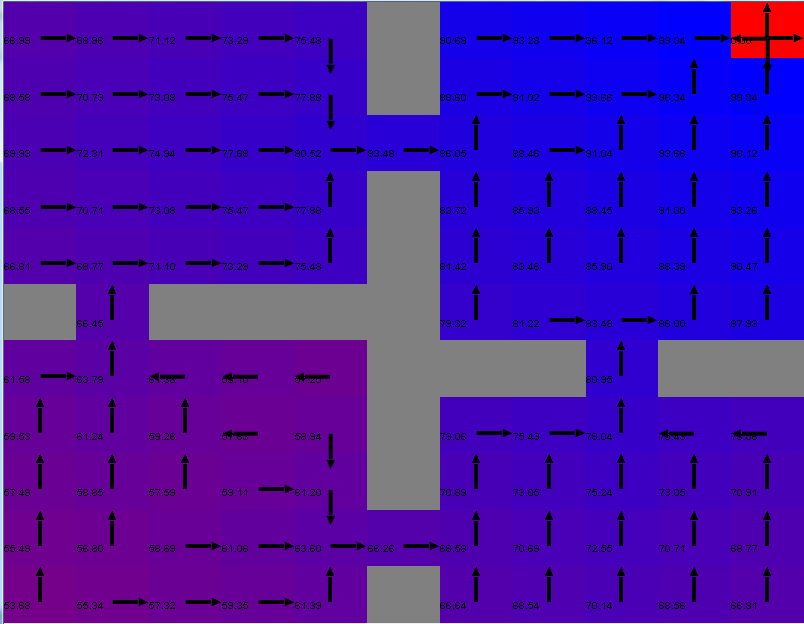


*Figure 3: Time vs gamma for PI and VI, Small grid*



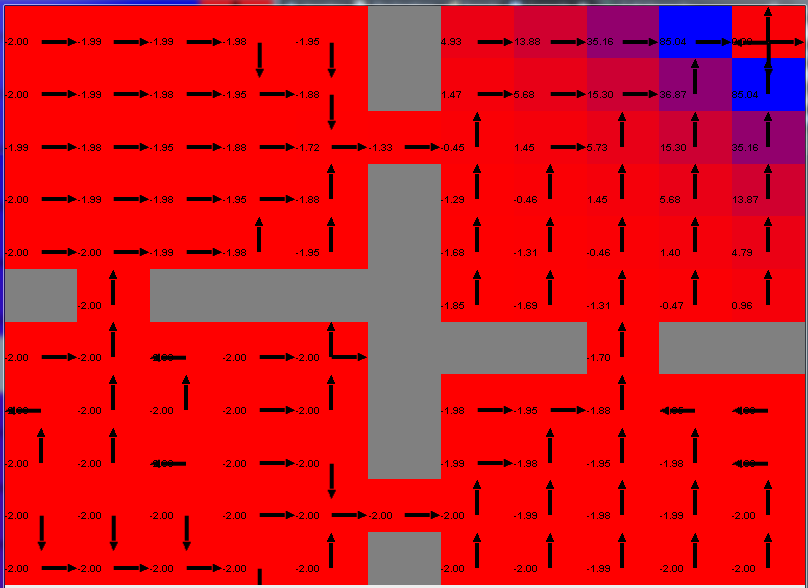
*Figure 4: Average Reward vs Gamma, Small grid world*

Figure 5 shows a policy map when gamma = 0.99. Here future reward is not being discounted much and thus the reward is able to propagate out from the goal state. The large gamma allows the agent to see the states closer to the goal, and is able to propagate the reward from those states to the earlier states. This creates a clear policy for the agent to follow. The utilities point clearly towards the goal state, with the states approaching the goal having incrementally higher utilities.



*Figure 5: VI, Policy map for gamma = 0.99*

Figure 6 shows a policy map when gamma = 0.5. Since future reward is discounted by such a large factor the utility is not able to propagate out from the goal state. The red coloring indicates that the utility for those states is very low. The utility of approaching the goal state is not realized until a few squares before the goal as shown by the blue in the top right corner. This is why some parts of the grid, especially in the bottom left corner in the start area, show the agent wandering in non-ideal directions. The agent does not know about the future reward because it is too far away and has been largely discounted. The utilities do not point toward the goal as clearly as in the case when gamma = 0.99.

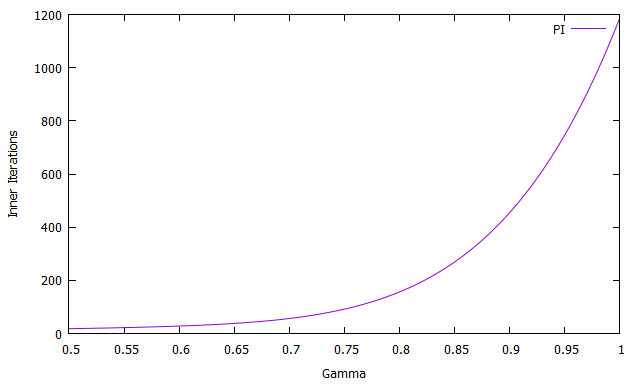


*Figure 6: VI, Policy map for gamma = 0.5*

**Policy Iteration**

PI is a planning algorithm that uses the model of the MDP to iteratively step towards the goal. The algorithm starts an iteration with a random policy and calculates the utility following that policy. The policy is then updated according to the action that maximizes the utility. The utility is calculated using Bellman’s equation. These steps are repeated until convergence. Rather than stepping through utility space like VI does, PI steps through policy space. PI typically takes fewer iterations to converge than VI. However, each iteration tends to be more expensive. Each iteration has an inner value iteration that computes a value based on actions defined by the current policy. The time complexity for each iteration of PI is O(A\*S2 + S3) where A is the total number of actions available from a state and S is the total number of states.

Figure 2 shows the iteration analysis for PI and VI. The number of iterations needed by PI when gamma = 0.99 is 6 compared to the 33 iterations needed by VI. The PI curve stays relatively level no matter the gamma choice with approximately 5 or 6 iterations being needed. Although PI requires less iterations than VI, these iterations tend to be more expensive because of the large amount of work being done within each iteration. Figure 7 shows the number of inner value iterations vs gamma for PI. The number of inner value iterations grows exponentially as gamma approaches 1. This increases the time needed to run PI.



*Figure 7: Inner value iterations vs gamma for PI, Small Grid world*

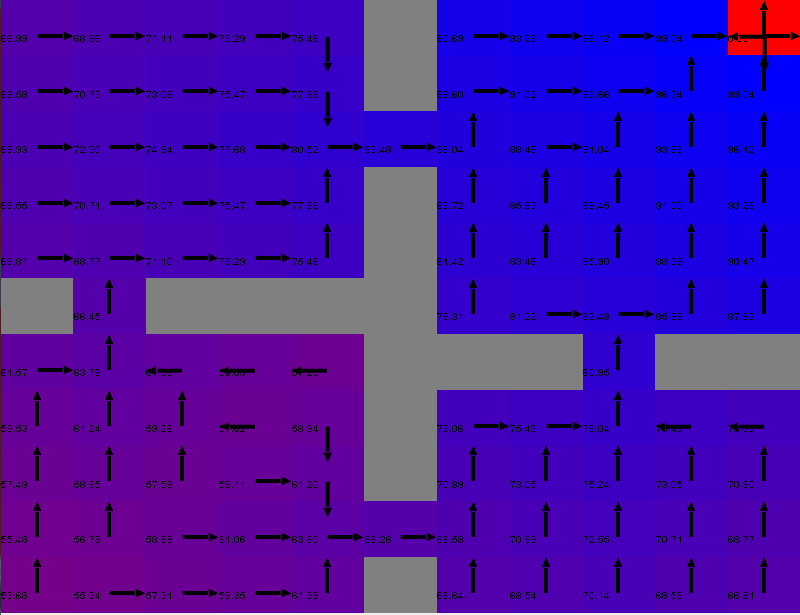
Figure 3 shows the time analysis for PI. PI is only able to outperform VI when 0.5 < gamma < 0.6. The time needed to converge grows exponentially as gamma approaches 1. Comparing Figure 3 to Figure 7 shows that the exponential increase is due to the increasing number of inner value iterations. The time needed when gamma = 0.99 is approximately 1500 ms compared to VI’s 52 ms. This is an increase by a factor of 25. Each calculation runs a Bellman update based on actions of the current policy. The Bellman updates are not as expensive as they were in VI. The system of equations being solved is now linear. Normal VI has a non-linear system of equations.

Figure 4 shows the reward vs gamma for PI compared to VI. The figure shows that PI achieves a nearly optimal reward for all gamma values. Since PI takes longer to run for larger gamma it makes sense to keep gamma small since we will receive the same reward. This graph also shows that PI does not need to look as far into the future as VI to achieve an optimal policy. Why? At gamma = 0.5 PI performs comparably to VI at gamma = 0.99 in reward received. It also takes less time to converge than VI with 34 ms compared to 52 ms.

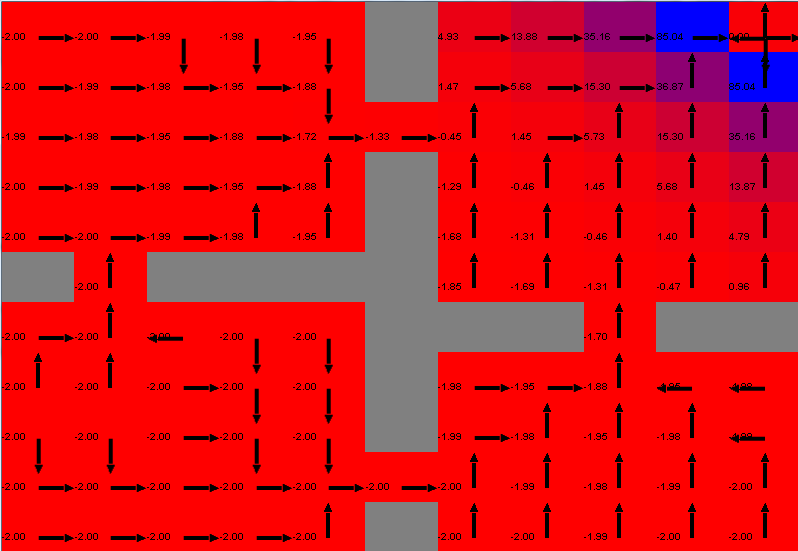
Figure 8 shows the policy map when gamma = 0.99. The policy is identical to the policy found by value iteration. Although VI finds the optimal policy, this does not necessarily mean VI found the true utilities. In fact, there are an infinite number of utilities that would result in the optimal policy. It is possible that the VI has found a set of utilities within this infinite set. However, upon closer inspection of Figure 5 and 8, the utilities found by VI and PI exactly match. This gives more evidence that the VI has in fact found the true utilities. The MDP is small enough that VI and PI can find the true utilities in a small amount of time.

Figure 9 shows the policy map when gamma = 0.5. Similar to VI, the small discount factor keeps the utilities from propagating out from the goal state. Thus the policy found is different from the policy found with a larger gamma. Although the policy is not optimal it still performs well. If we choose a small gamma, PI is able to outperform VI in the small grid world with respect to time, while achieving the same reward.

Show the average time per iteration for VI and PI. PI iterations take longer and thus have a higher average. However, each iteration takes less time than the previous because it requires less value evaluations.



*Figure 8: PI, Policy map for gamma = 0.99*



*Figure 9: PI, Policy map for gamma = 0.5*

**SARSA**

SARSA belongs to the family of Q-learning algorithms. In contrast to other Q-learning algorithms, when choosing the next state to move to, the SARSA algorithm does not necessarily choose the state with the highest Q value. Instead it follows the policy that determined the original action and applies it to the new action. This allows the algorithm to iteratively improve the policy every episode by taking into account the previous policy. It also for every state will take, with some probability, a random exploration step rather than following the policy. SARSA is an on-policy learner. This means that while the agent is exploring, SARSA learns the cumulative reward of the policy being learned. This is in contrast to the standard Q learner which is an off policy learner. It does not depend on the agent’s actions to learn the optimal policy. SARSA may find a different policy for problems where Q-learning may accumulate large penalties for exploring.

SARSA’s time performance is plotted with VI and PI in Figure 3. SARSA takes much more time than VI and PI to converge. The time for convergence decreases as gamma increases. At gamma = 0.5 SARSA takes approximately 4500 ms. At gamma = 0.99 the time is approximately 2600 ms. The decrease in time with respect to increasing gamma is in contrast to PI and VI. Both VI and PI’s time increased with increasing gamma. The reason for the difference is the alternate strategies used for exploration. VI and PI do not need to explore as they already have a full model of the MDP. SARSA, on the other hand, does not have a model and must spend much of its time exploring. For small discount factors the agent does not have as much information about the future reward and thus will tend to wander randomly with less guidance towards the goal. For large discount factors the agent now has more information about the large future reward and does not need to explore as much, thus decreasing the time needed.

Figure 10 shows a comparison of two different learning rates for the SARSA algorithm. For this figure gamma was set to 0.99 since that was shown to have the best time performance. The blue line indicates a learning rate (L) of 0.9 and the red line indicates a learning rate of 0.1. Both curves use an initial Q of 0. The x-axis for each sub plot represents the episode. A total of 100 episodes was used for the algorithm.

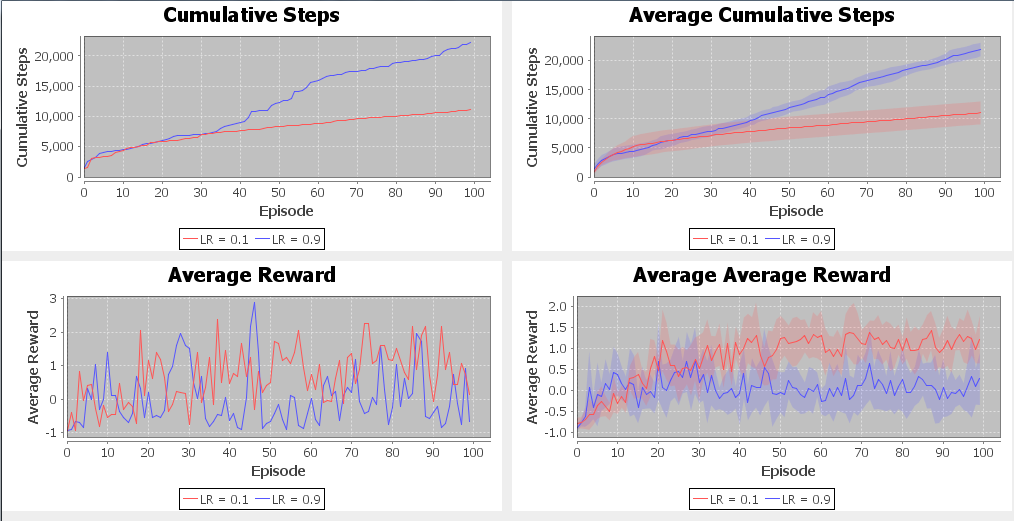
Both curves initially start out performing similarly. At episode 30 the L = 0.9 curve begins to diverge and the number of steps needed begins to increase much more rapidly. By episode 100 the number of cumulative steps has increased to 22,000 compared to just 11,000 for the L = 0.1 curve. Comparing the average average reward plot in the bottom right corner of Figure 10 it can be seen that the gained reward also flattens out at around episode 30. By episode 30 the agent starts approaching a limit on the amount of reward that can be gained. The gains beyond episode 30 are minimal. Increasing the number of episodes to 10,000 only slightly improves performance because the agent has made most its gains by episode 30. However, the agent with a high learning rate is still trying to take in new information beyond episode 30 instead of exploiting the old. It searches more intensely for new information while the agent with a low learning rate is now exploiting the knowledge it has gained.

The average reward plots in Figure 10 show that the SARSA algorithm did not perform well on this problem. The VI and PI were able to achieve max rewards of 75. SARSA was only able to achieve a max reward of 3 in the best case. The positive value indicates that the agent is able to reach the goal. However, the fact that the value is so low shows that the agent is taking a very non-optimal route to get there. SARSA’s exploration strategy could improve the performance if given enough time. However, there is a large number of ways to approach the goal state. It is, therefore, very time consuming the SARSA algorithm will ever find the optimal path. Overall Figure 10 shows that a low learning rate outperforms a high learning rate. Additional L values were experimented with however, decreasing L further below 0.1 only improved performance slightly.

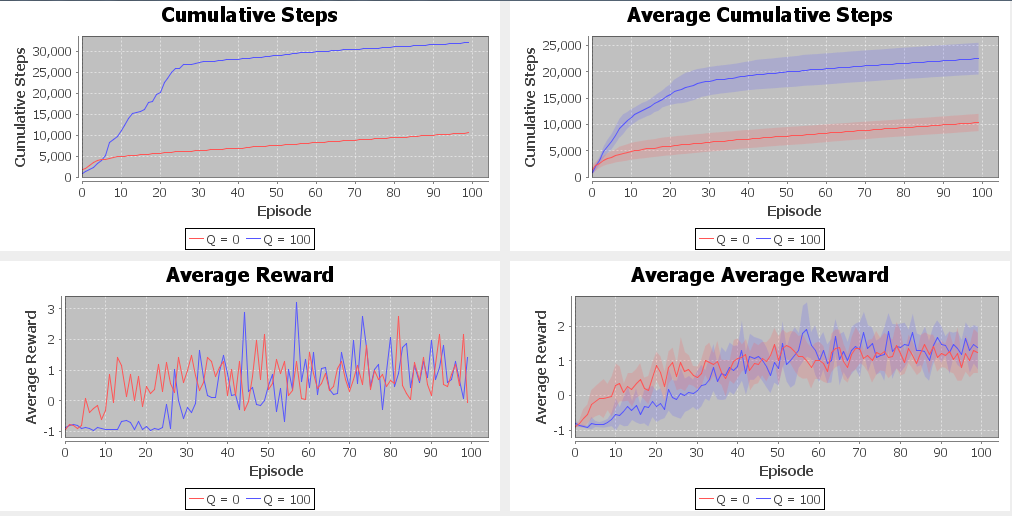
When initializing the SARSA algorithm the programmer is given a choice to set an initial Q value for all the states. Figure 11 shows a comparison of two different initial Q values for SARSA. For both curves L = 0.1 was used since that was shown to perform better previously. For this figure gamma = 0.99 was used since that was shown to have the best time performance. The red line indicates an initial Q of 0 and the blue line indicates an initial Q of 100. The cumulative steps portion of the plot shows a strong difference in the performance of the two agents. For the Q = 0 curve there is a steep increase in number of steps from episodes 0 to 5 with approximately 50% of the total accumulated steps occurring in this region. For these first 5 episodes the agent is more strongly exploring the state space. After episode 5 the agent has discovered the goal and spends less time exploring. This can be seen in the average reward plots. Around episode 5 the reward becomes positive indicating the agent has reached the goal state. After the agent discovers the goal state it spends less time exploring random policies and instead focuses on improving the policy that led to the goal. Therefore, fewer steps are accumulated after episode 5. The number of steps continues to increase but at a slower rate. After 100 episodes the total number of steps is 10,000.

The agent with initial Q values of 100 does not perform was well as the agent with a Q initialization of 0. The total number of steps required is 30,000 compared to the previous agent’s 10,000. It also takes many more episodes to initially reach the goal state. The average reward plots of Figure 11 show the reward does not become positive until episode 25. Similar to the previous agent the cumulative steps sharply increase until episode 25 because the agent is still strongly focused on exploring. Overall the two different starting parameters cause the algorithm to converge to a similar policy which is obtaining the same average reward. However, with Q=100 SARSA takes much longer to converge.

Overall, SARSA’s performance is weak compared to VI and PI. The weakness stems from the fact that agent has no model to work with and instead must discover the transition function and reward function on its own. SARSA was also compared to a Q-learning algorithm that uses a greedy approach rather than an exploratory approach. The greedy Q-learning algorithm came up with a similar max reward as SARSA; however it was able to achieve the reward in a shorter amount of time. SARSA is a better fit for an environment where continuous exploring is necessary. However, the grid world used is stationary. Once the goal state is found with an optimal policy there is no need for further exploration because the optimal policy will never change. Although in this case SARSA was not able to find the optimal policy.



*Figure 10: Comparison for different learning rates for SARSA*



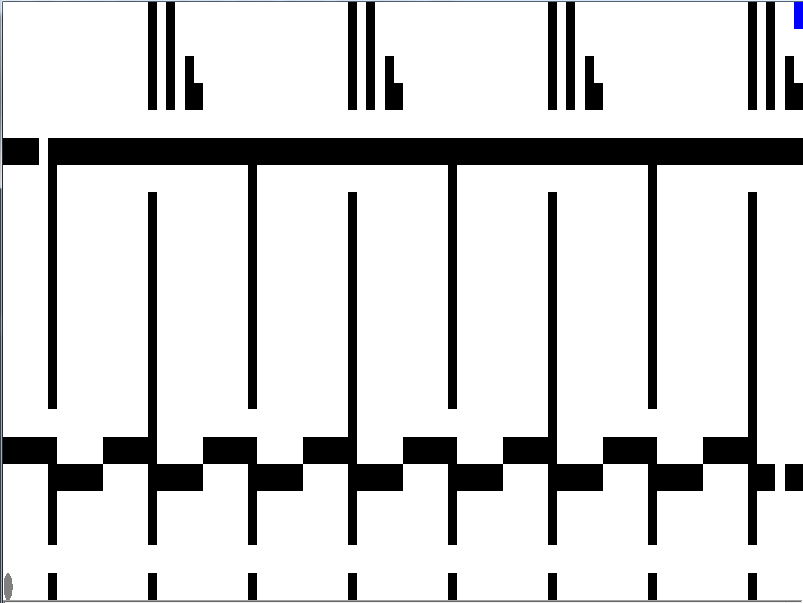
*Figure 11: Comparison for different initial Q values for SARSA*

**Large Grid World**

The large MDP for this assignment is an 87 x 21 grid with 1,614 states. The large grid world is shown in Figure 12. The agent starts in the bottom left corner and is trying to reach the goal state in the top right corner marked by the blue square. In the middle portion the agent must make its way through a winding section. Similar to the small grid world, the reward for entering a non-goal state is -1. However, in this case the reward for entering the absorbing goal state is 30,000.

***Why is it Interesting?***

The large grid world gives us a basis for understanding the behavior of the different algorithms when a large number of states are introduced. The increased scale will prove to be difficult for the SARSA and PI algorithms, while showing the strengths of VI. The winding of the middle section also creates a more difficult path for the agent to discover as the optimal policy resulting in increases in algorithm running times and iterations. The winding section also decreases the reward since the agent takes more steps to reach the goal than if the section were just a straight pass.



*Figure 12: Large grid world*

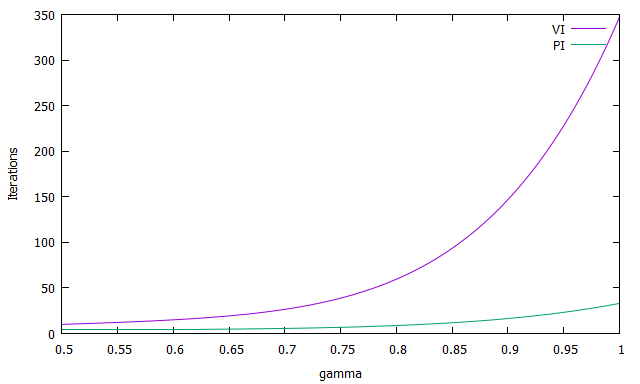
**Value Iteration**

Figure 13 shows the iterations vs gamma plot for both VI and PI. At gamma = 0.5 VI and PI have roughly the same amount of iterations. As gamma approaches 1 the iterations diverge. PI’s iteration increase only slightly from 10 to 40. With VI, however, the iterations increase exponentially from 15 to 350. For both PI and VI the number of iterations increased 10-fold from the small grid world while the number of states grew by a factor of 15 between the two worlds. Therefore, the growth in number of iterations with respect to number of states is approximately linear.

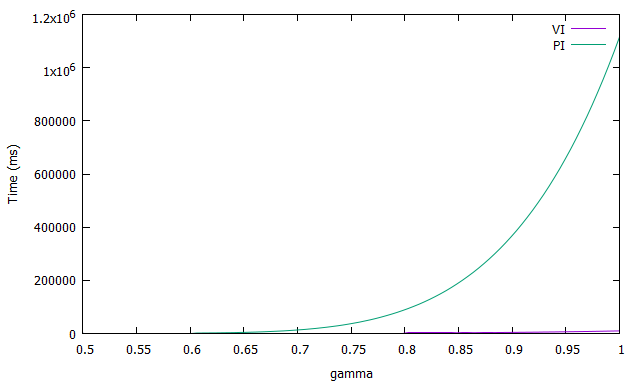
Figure 14 shows the time vs gamma plot for both VI and PI. The plot has similar characteristics to the time plot from the small grid world. Both VI and PI have similar time performance for smaller gammas. As gamma approaches one the VI and PI curves diverge. The required time for VI to converge grows only slightly with increasing gamma while PI grows exponentially. At gamma = 0.99 VI takes approximately 9.7 seconds and PI takes 1114.7 seconds. PI takes about 123 times longer than VI. Comparing VI’s 52 ms for convergence at gamma = 0.99 for the small grid world it can be seen that VI’s time for convergence has increased by 186 times in the large grid world. This increase differs from the increase in number of iterations between problem sizes. The number of iterations required appeared to grow linearly with the number of states. The time, however, appears to be growing exponentially. The large increase in time is due to the increase in computations for Bellman updates. Each additional state that is added must now perform additional recursion computations for all the actions it can take to neighboring states. Therefore for each additional state that is added, an even larger number of computations must be performed to find the utility for that state, increasing the time exponentially.

Figure 15 shows the rewards vs gamma plot for VI and PI. The figure shows the rewards for gamma values between 0.98 and 0.99. The reason for this small range of gamma is because for gamma < 0.98 VI was not able to produce a policy that allowed the agent to reach the goal. The resulting policy from gamma values less than 0.98 caused the agent to wander endlessly, accumulating negative rewards, until the computer ran out of memory. The necessity of a large gamma is one way in which the large grid world differs from the small grid world. Before, in the small grid world, even a small discount factor the VI still produced a policy that led the agent to the goal. Now in the large grid world the size of the problem prohibits the use of a small discount factor. Before, in the small grid world, with the smaller discount factors, the utilities were already reaching a propagation limit from the goal state to the start state as was shown in Figure 6. In Figure 6 the three rooms near the goal state had optimal policies for all their states. The far room, however, was already showing signs that it was not receiving optimal utility updates. It was shown that there were a few non-optimal states where the agent would wander near the starting area. Eventually the agent was able to escape into a state that would lead it through states pointing to the goal state. Now the space is too large for any meaningful utilities to propagate from the goal state to the start state for small gammas. The result of a small gamma is a fairly random policy surrounding the starting area where the agent gets stuck wandering. Since the MDP keeps no memory of previous actions, the agent will have no idea it has been wandering around the same area. Its only hope to escape is an unlikely sequence of accidental missteps, opposing the specified policy, towards the goal until it reaches a state near the goal that has updated utilities.

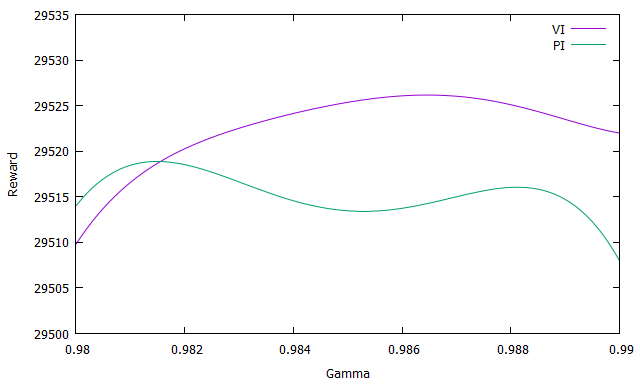
In the large grid world if the agent were to follow the most optimal policy perfectly without any missteps it would take 390 steps accumulating a reward of -390 before it reached the reward of 30,000. This would result in a max reward of 29,610. Figure 15 shows VI is coming very close to achieving the max reward. At gamma = 0.986 VI achieves a reward of 29,525. The policy found by VI is very likely to be close to the optimal policy as the agent is only 85 points away from achieving the maximum reward. The loss of 85 points is most likely due to the probabilistic missteps of the agent while following the optimal policy towards the goal. However, since the policy is optimal or near optimal it allows the agent to gracefully recover from any missteps and quickly gets the agent back on track towards the goal state. The goal state is less likely to be reached without probabilistic missteps because the space is larger and requires more steps before the game is complete. Therefore, it is important that the policy quickly gets the agent moving towards the goal state after a misstep in this large space.



*Figure 13: Iterations vs. gamma, Large grid world*



*Figure 14: Time vs gamma, Large grid world*



*Figure 15: Reward vs gamma, Large grid world*

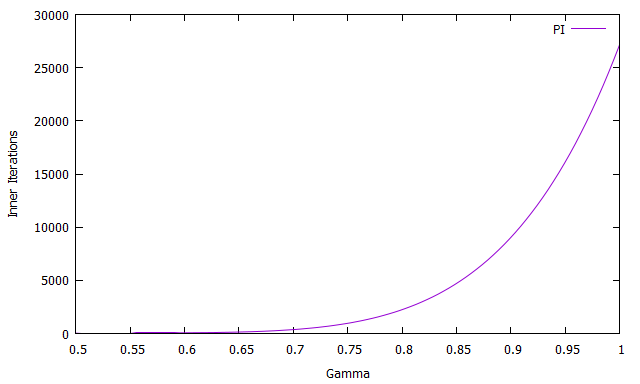
**Policy Iteration**

Figure 13 shows the number of iterations required for PI to converge for different values of gamma. The plot represents the number of outer policy iterations. Similar to the case of the small grid, the PI requires many fewer iterations than VI at large gamma values. Unlike the small grid world, however, the number of required PI iterations does not stay relatively constant as gamma increases. The number of iterations starts at about 5 for gamma = 0.5 and increases to 40 for gamma = 0.99. The increase in iterations for increasing gamma is caused by the way gamma propagates utilities in a large space. For small gammas the number of inner value iterations is very small. The planner is not looking very many states ahead and thinks it has optimized after very few inner value iterations, thus moving on to the next outer iteration. For large gammas the planner is looking many states ahead and takes many inner value iterations to update the policy, often maxing out and being forced into the next outer iteration. This happens because, for each additional state added to the space, the inner value iteration must check all of the neighboring states when performing a Bellman update. For the small grid even for large gamma the PI did not need to increase the inner value iterations by much because there are not that many extra states to consider. In addition, for the small grid the PI was able to find a strong policy even with a low gamma. For the large grid the PI was not able to find a working policy for small gamma values.

PI is guaranteed to converge because the value iterations are always improving our estimate of the policy and there are a finite number of policies. In the case with small gammas PI is converging to a local optimum and finding a weak policy. This can be seen by the small number of inner value iterations. For small gammas there are many fewer inner value iterations that there are at large gammas.

Figure 14 shows the required running time for PI for different gamma values. PI starts out with an extremely low running time for low gammas. This is due to the small amount of inner value iterations and outer iterations as shown in Figure 16. At about gamma = 0.7 the time begins to increase exponentially with respect to gamma. The increase in time correlates with the increase in inner value iterations in Figure 16. The exponential increase in time with respect to gamma behavior for the large grid world is similar to the behavior of PI in the small grid world. It is interesting to note the PI in both worlds’ sizes takes approximately the same time to converge for small gammas. This is because, regardless of the world size, for a small gamma the PI will consider roughly the same number of states when looking ahead. Since the large reward is so far away and cannot be seen by PI the algorithm thinks there are no utility updates to perform and completes the updates in roughly the same number of inner value iterations. Comparing Figures 7 and 16 shows that the number of inner value iterations is roughly the same for both problem sizes.

The time difference for convergence between the two worlds begins to show at large gammas. For gamma = 0.99 the time needed for convergence is 1114.7 seconds. Compared to the 1500 ms needed in the small grid world the PI now takes approximately 700 times longer to converge. Similar to the VI, the PI needs an exponentially larger amount of time when the number of states increases linearly. However, the PI’s exponential increase is much sharper, as VI’s running time increased by a factor of 100. This result is not surprising considering the PI needed a much larger number of iterations to converge and the time for each iteration is very computationally expensive. Comparing Figures 7 and 16, it can be seen that PI requires about 30 times more value iterations for the large grid world than the small grid world. Clearly the increase in number of inner value iterations does not fully account for the large growth in time requirement. The increase in time is also due to the larger amount of time taken by each inner value iteration. The major difference between the inner value iteration in PI and the original VI algorithm is the linearity of the equations in the inner value iteration for PI. The VI algorithm, on the other hand, has non-linear Bellman update equations. Generally, the linearity of the value updates in the PI algorithm speeds up the inner value iterations by allowing matrix inversions. However, as the number of states grows the size of the matrix also grows and eventually the matrix inversion becomes computationally expensive. The expense of inverting the value equation matrix is greatly increasing the computation time of PI in the large grid world.



*Figure 16: Inner value iterations vs gamma, Large grid world*

Figure 15 shows the reward payoffs for different gamma values for PI. Similar to VI, the PI could not find a working policy for gamma < 0.98. The policy found at lower gammas caused the agent to wander endlessly. PI shares this weakness with VI because within every policy iteration there is an inner value iteration. Thus, the inability of the VI to look far enough through the MDP also hinders the PI.

Many of the utilities in the starting sections are the same. Thus the agent is taking random actions to break the ties between states. Ultimately the agent just wanders near the starting area. The figure shows that overall PI does not perform as well as VI. Even at large gamma values, 0.99, the policy found by PI nets the agent a reward of 29,508, a difference of 14 points from the VI at that same gamma. The policy found by PI is leading the agent through non-ideal states and causing it to accumulate unnecessary negative reward.

Earlier in the small grid we were able to use PI with a smaller gamma to find an optimal policy. For the large grid a working policy cannot be found if gamma < 0.98, forcing gamma to be large. The large gamma, in turn, means the time requirements for PI will be very high. Since we are forced to use the higher gamma we are stuck with a slow convergence. For gamma = 0.99 PI takes approximately 100 times longer than VI to find a policy. In addition the reward accumulated by the PI policy is less than the reward accumulated by the VI policy. VI outperforms PI in both time and reward accumulated and thus is better suited for the large grid world than PI.

**SARSA**

Figure 15 shows the results for different learning rates applied to SARSA. For these plots the initial Q values are all zero and the discount factor = 0.99. The red line indicates a learning rate of 0.1 and the blue line indicates a learning rate of 0.9. The x-axis for each sub plot represents the number of episodes. SARSA was run for 20 episodes to generate these plots in contrast to the 100 episodes used for the small grid world. The reason for this difference is the increased amount of running time of SARSA for the large grid world.

In the top left corner of Figure 15 the steps accumulated over each episode is shown. For L = 0.1 the total steps for 20 episodes is approximately 3,000,000. The figure shows that 66% of the steps are accumulated before episode 8. For L = 0.9 the total cumulative steps at episode 20 are approximately 750,000. These results show that the higher learning rate is superior to the low learning rate. This result is the opposite of the small grid world where we found smaller learning rates performed well. For such a large number of states the learner requires a large amount of data about the future rather than preserving its original knowledge, thus it needs to weigh future values more heavily than current values. Another difference between the small and large grid worlds is the increase in number of cumulative steps. In the best case for the small grid world the total accumulated steps was 11,000. The best case here for the large grid world has 750,000 steps which is about a 70-fold increase. Although the number of states has increased linearly, the number of steps required has increased exponentially.

The bottom left plot of Figure 15 shows the average reward obtained for each episode. The average reward starts off at -1 for episode 1 and increases to 4 by episode 20 for L = 0.9. For L = 0.1 the reward hovers around -1 for all the episodes. These results show a sharp contrast between the planning algorithms, VI and PI, and the learning algorithm SARSA. Both PI and VI managed to achieve a reward above 29,500 while SARSA only managed to achieve a reward of 4. The low reward is due to the algorithm’s need to wander around to learn about its environment. In contrast, the planning algorithms are provided with a model and do not need to explore the environment.

The average reward plot also shows the L = 0.9 curve sharply increasing around episode 20. If the trend continues it is possible SARSA would perform better given many more episodes. However, the time necessary to run the extra episodes still makes the SARSA an inefficient algorithm for this problem. Increasing L is another possible way to improve performance. Additional experiments increasing L closer to 1 were run, but performance improved only slightly.

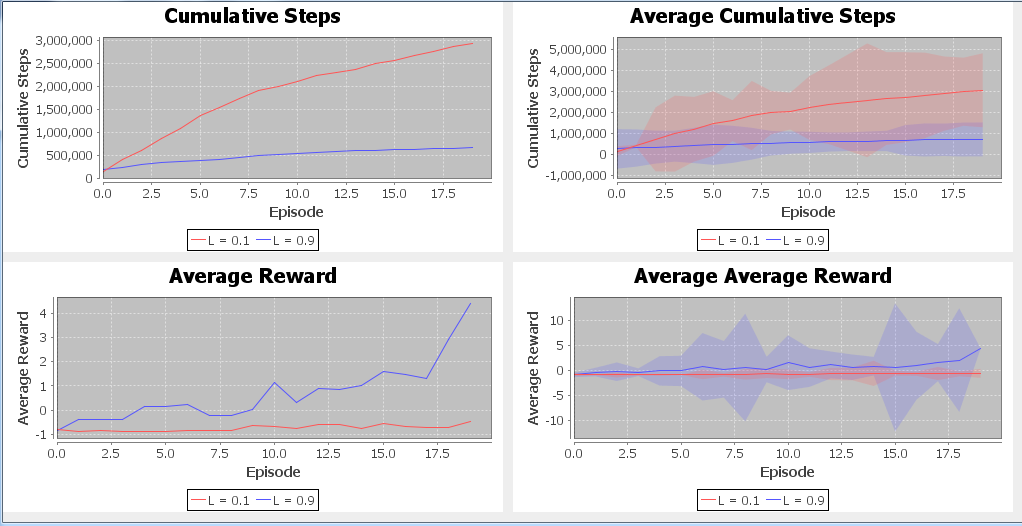
Figure 16 shows the results for different initial Q values. The plot uses a learning rate of 0.9 and a gamma of 0.99 since those parameters proved to perform better in earlier trials. The red line indicates an initial Q value of zero. The blue line indicates an initial Q value of 30,000. The number of cumulative steps for each curve is similar. Both curves have a sharp initial increase. At episode 3 both curves have accumulated about 400,000 steps, a little more than 50% of the total accumulated steps. At episode 20 the Q = 0 curve has accumulated 600,000 steps and the Q = 30,000 curve has accumulated 750,000 steps. The cumulative step and average reward curves start to diverge at episode 13. The agent with initial Q = 0 has reached the goal state and therefore spends less time exploring. The agent with initial Q = 30,000, however, has not reached the goal state and continues to explore and accumulate a large amount of steps every episode.

The bottom left plot of Figure 16 shows the average reward. Here we can see that for Q = 30,000 the reward hovers around 0. For Q =0 the average reward starts to increase dramatically at episode 15. This indicates that if we used more episodes we could increase the reward the agent receives. However, time constraints prohibit this. The bottom right plot in Figure 16 also indicates the variance for the different trials. At episode 17 it can be seen that for one of the trials the max reward reaches 40. On the other hand, another trial reaches a low reward of negative 30. Clearly there is a large amount of variance in the trials.

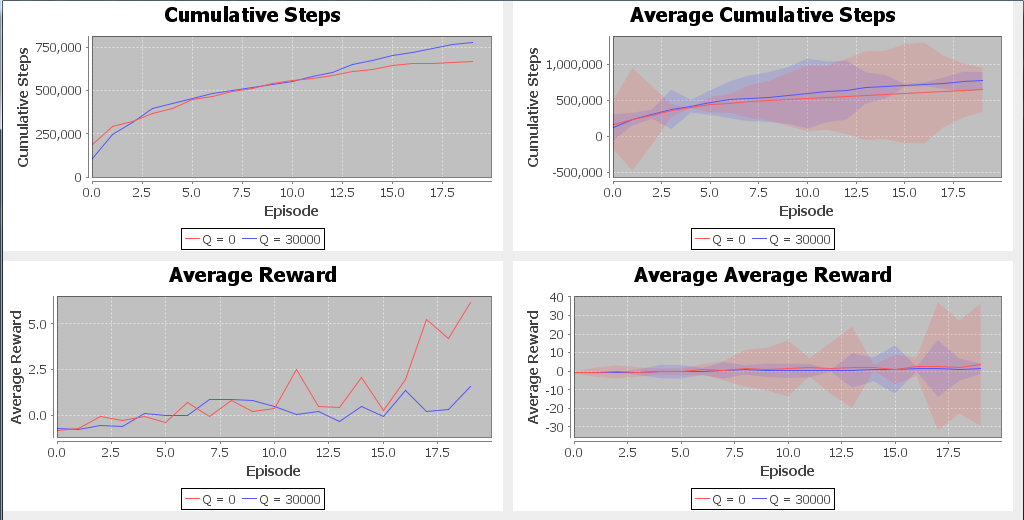
The running times for SARSA varied based on the learning rate. The reason for the variation is the different number of steps required. For this reason the running times for the changing initial Q values did not vary much because they have roughly the same number of steps. For L = 0.9 SARSA took approximately 915.4 seconds to complete the 20 episodes. Each step takes approximately 1.52 ms. For L = 0.1 SARSA took approximately 4560 seconds to run 20 episodes, about 5 times longer than the L = 0.9 case. This makes sense because there are about five times as many steps in the L = 0.1 case. The time taken for each step is the same as for the case of L = 0.9.

The running time for SARSA in the large grid world was 1.4266288E7 milliseconds or approximately 3.96 hours (this should be split in quarter, currently this is for running two algs for two trials) which prohibited it from being displayed on the figure with the other algorithms.

The second figure took 3661569.0 milliseconds.



*Figure 15: Comparison for different learning rates for SARSA*



*Figure 16: Comparison for different initial Q values for SARSA*

**Conclusion**

The strengths and weaknesses of VI, PI, and SARSA were explored using a small and large domain. In the small domain PI was shown to be the superior algorithm. It was able to achieve an optimal policy in less time than VI by exploiting the short running times for small gammas. Overall VI performed well and was able to find an optimal policy but took slightly longer than the PI. SARSA did not perform well in the small grid world. Not only did it have the longest running times, but it failed to achieve a policy that resulted in high rewards.

In the large grid world VI was shown to be the superior algorithm. It was able to achieve a high reward and also had the shortest running time. It greatly outmatched PI in terms of running time. PI was able to find a policy that achieved a high reward. However, the reward was a bit lower than that achieved by VI. The main drawback of PI in the large domain was the large running time. SARSA, similar to the small grid world, was the weakest performer both in terms of running time and reward accumulated.

The weakness of SARSA in both environments points to a greater distinction between planners and learners overall. A learner can never really outperform a planner. This is because a planner allows a programmer to insert domain knowledge about the world. A learner, on the other hand, does not have any domain knowledge. It must learn for itself. If possible, it would be preferred that a teacher would always be to insert domain knowledge and use a planner. This would greatly increase efficiency. However, it is not always possible for a planner to be used. There are non-stationary environments. With the constant change, it is not feasible for a programmer to be continuously inserting domain knowledge into a planner. Instead a learner will be used that can constantly update its policy without human intervention. There are also environments where a learner may be able to teach the programmer some domain knowledge based on the policy it finds.