Markov Decision Processes

Gary Saavedra

gsaavedra3@gatech.edu

Introduction

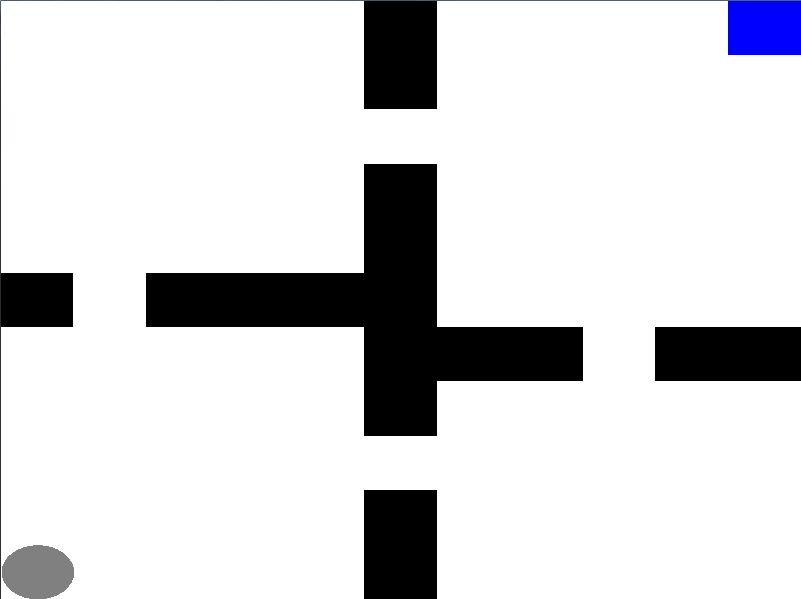
There are two types of algorithms that can be deployed, a planner and a learner. A planner can be used when there is a model for the MDP the agent is trying to move through. Specifically there must be a transition function and a reward function. A learner can learn a policy even when there is no model. 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 Markov Decision Processes (MDP), a small grid and a large grid world were used.

Small Grid World

A small 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, 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 goal state. Otherwise, the agent receives a reward of -1 for entering any other state. This gives the agent motivation to end the game as quickly as possible.

*Why is it interesting?*

The small grid world will show the strength of the planning and learning algorithms. It allows for an analysis over many trials because there is not as much time needed.



*Figure 1: Small Grid World*

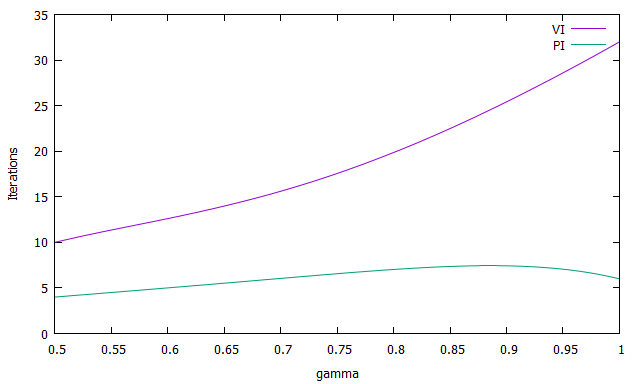
Value Iteration

VI is a planning algorithm that utilizes the model from 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 and so on. For VI 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. (See GreedyQPolicy in ValueIteration for more in burlap).

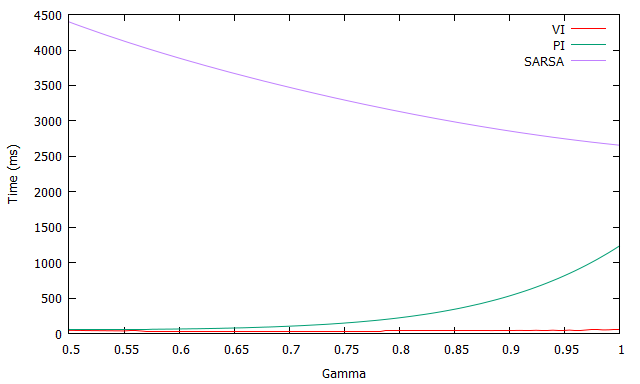
Figure 2 shows an iterations vs gamma graph for both PI and VI. Figure 2 represents the number of iterations needed for convergence for a specified gamma. Gamma throughout this paper represents the discount factor. Figure 2 shows the VI always has a larger number of iterations no matter the choice of 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 to perform by 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. Beginning at gamma = 0.6 the agent is considering future reward enough that it tends to not wander.

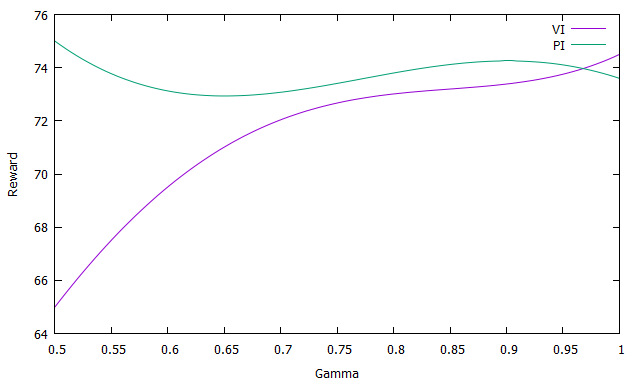
Figure 4 shows a plot of rewards vs gamma for VI. The highest reward of approximately 75 is achieved when gamma = 0.99. VI comes very close to converging to an optimal answer beginning at gamma = 0.7. In an ideal situation if VI is able to get to the goal without accidentally probabilistically taking a wrong direction it can get there in 21 steps which would result in a reward of 79. Given the stochastic nature of the grid the VI has likely converged on the global optimum of 75. If time were a factor we could instead use a gamma of 0.7 and converge to a nearly optimal answer while saving time. Comparing Figures 3 and 4 we can see that although VI is finding a policy is 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 far enough into the future to estimate utilities the states near the starting state are not updated optimally as shown in Figure 6. This causes 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 negative ones resulting in a lower final reward.



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

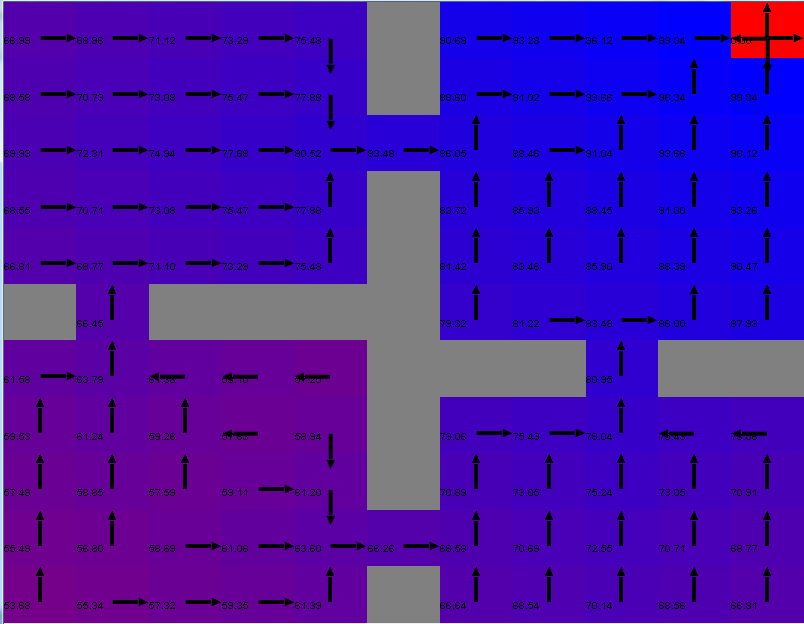


*Figure 3: Time vs gamma for PI and VI, Small grid*



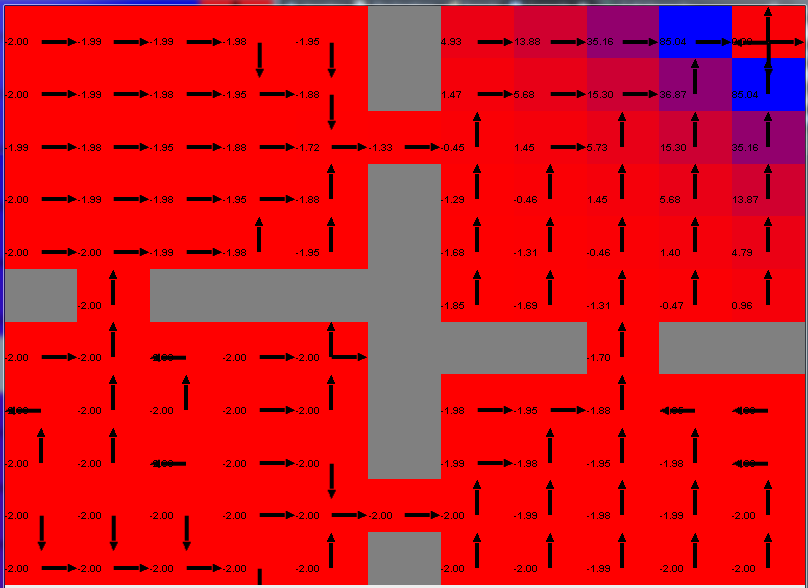
*Figure 4: Average Reward vs Gamma, Small grid world*

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 agent looks further into the future, 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.



*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 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 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 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 less 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. There is a tradeoff between PI and VI. Although PI takes less iterations to converge than VI, the iterations are much more expensive. In order for PI to converge in less time that VI, it must have many fewer iterations.

For both PI and VI number of iterations increases as gamma increases.

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 either 5 or 6 iterations being needed. PI takes many fewer iterations than VI. However, these iterations tend to be more time expensive because of the large amount of work being done. Each iteration has several, possibly hundreds, of value iteration calculations. 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 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. Why does this happen? How is discount factor affecting this? 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. 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 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.

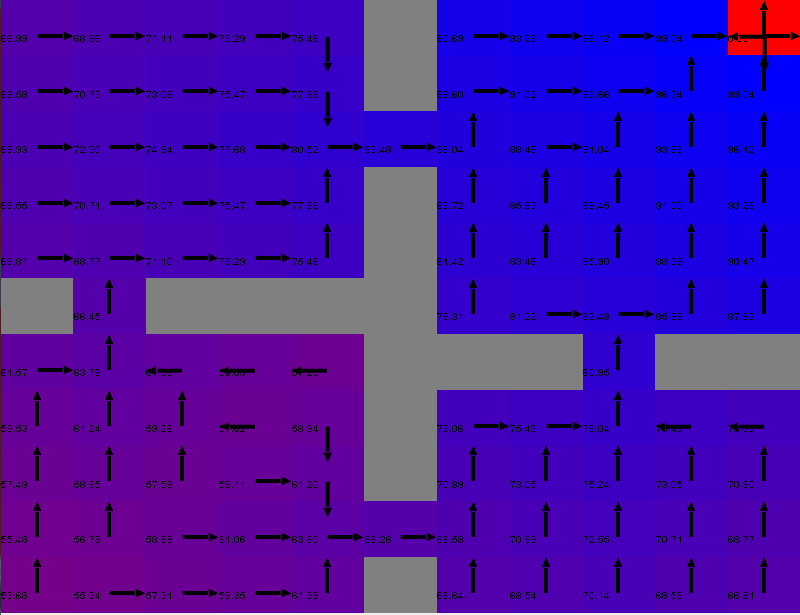
Figure 7 shows the policy map when gamma = 0.99. The policy is identical to the policy found by value iteration.

Figure 8 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. The policy found is different from the policy found with a larger gamma.

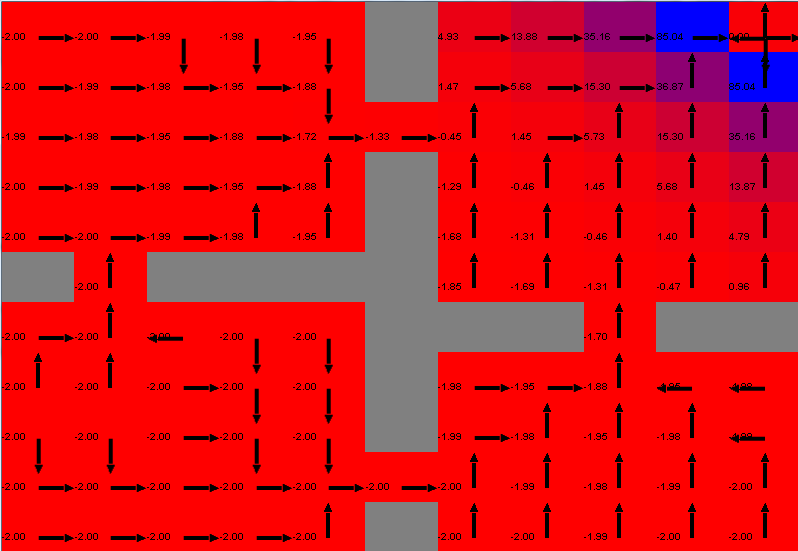
VI and PI converge to the same policy. This means either the VI found the true utilities or utilities that created the correct sequence of actions that led to the policy.

Explain why I’m using gamma = 0.99 for all my choices.

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 7: PI, Policy map for gamma = 0.99*



*Figure 8: PI, Policy map for gamma = 0.5*

**SARSA**

SARSA belongs to the family of Q-learning algorithms. It differs by the fact that when choosing the next state to move to it 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.

SARSA is an on policy learner. This means that while the agent is exploring SARSA learns the cumulative reward of the policy being learned. It also for every state will take with some probability a random exploration step rather than following the max Q value. This allows the algorithm to iteratively improve the policy every episode by taking into account the previous policy. 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. For conclusion: SARSA was not able to outperform Q-learning in this case because a large amount of exploration is not needed to reach the goal state. Often SARSA came up with a similar policy to Q-learning but took more time to find it. SARSA is a better fit for an environment where continuous exploring is necessary. However, the large grid world used is static. Once the goal state is found with an optimal policy there is no need for further exploration because the optimal policy will never change.

Figure … shows a comparison of two different learning rates for the SARSA algorithm. The blue line indicates a learning rate of 0.9 and the red line indicates a learning rate of 0.1. Both curves use an initial Q of 0.

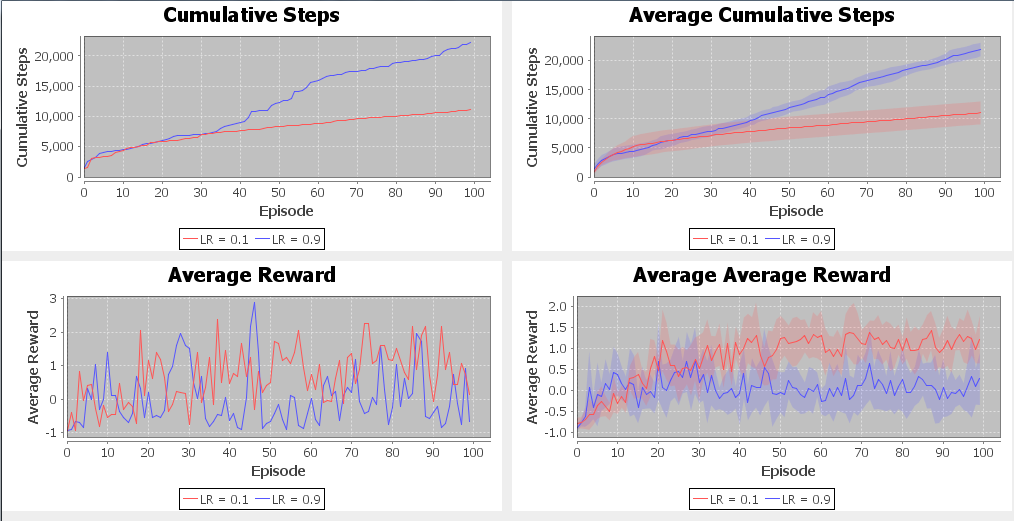
Iteration analysis is not apples to apples with the other algorithms. This is because the agent is now moving through the environment to learn about it. An iteration therefore is one movement from the agent or a whole episode?

Figure … shows a comparison of two different initial Q values for SARSA. The red line indicates an initial Q of 0 and the blue line indicates an initial Q of 100. 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.

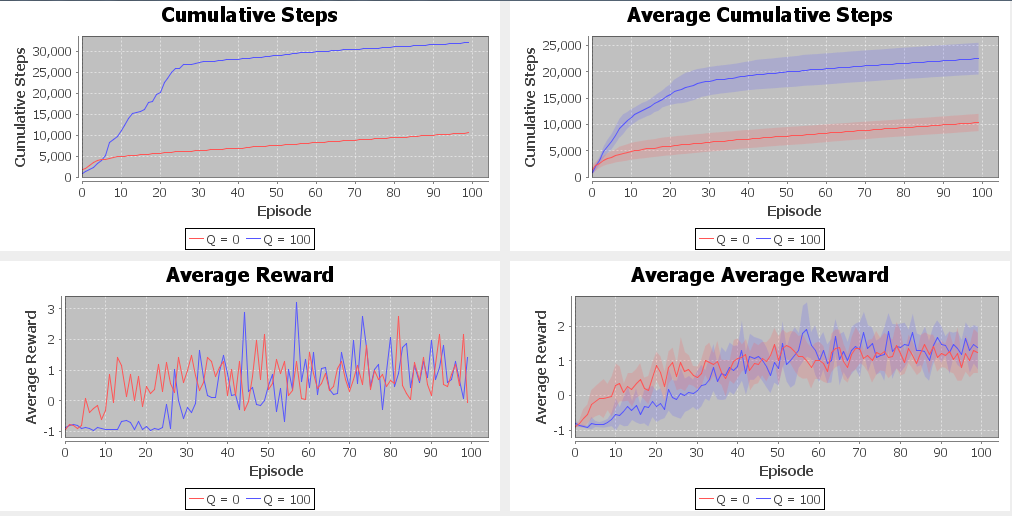
SARSA takes less time for larger gammas because it can see into the future. It does not wander around the initial stages as much.

If you have space talk about how the steps change from episode to episode. First few episodes have a lot of steps and afterwards the number of steps kinda randomly osciallates amongst much lower numbers. Can kinda be seen in the av cum steps graph with the initial sharp increase.

Does reward for each episode add to all other rewards for each episode so that it’s cumulative. That would partly explain the poor performance



*Figure : Comparison for different learning rates for SARSA*



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

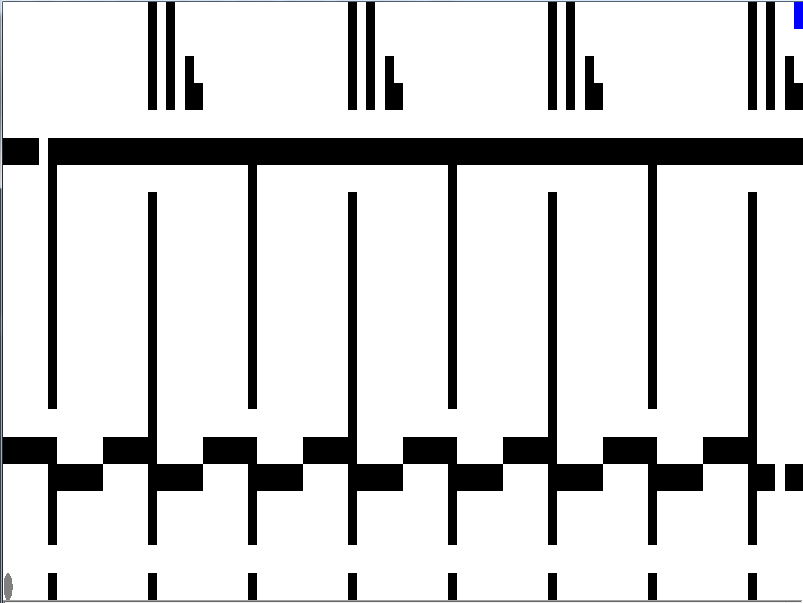
**Large Grid World**

***Why is it Interesting?***

The large grid world is shown in Figure… 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. There are 1614 states.

In the middle portion the agent must make its way through a winding section. The agent takes significantly more time through the winding section than when the section was just a straight travel through.

Talk about rewards fir reaching goal state etc.

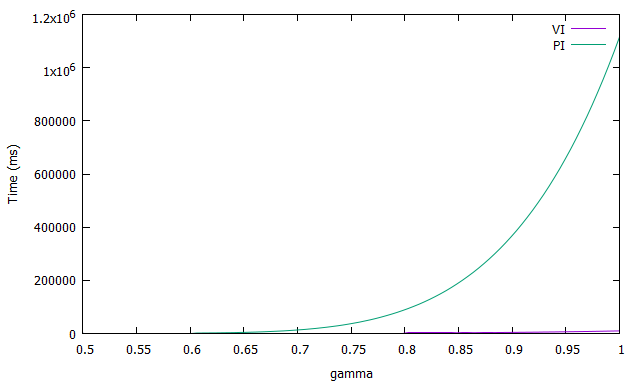


*Figure : Large grid world*

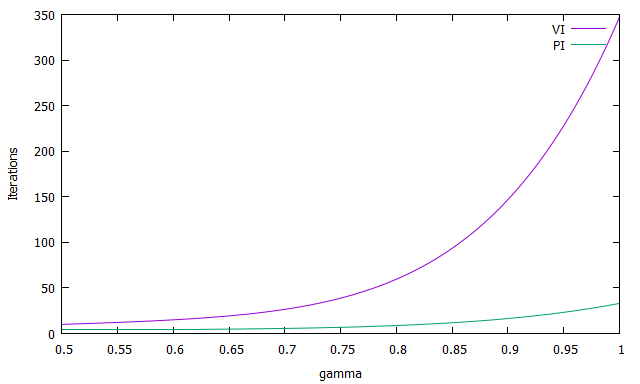
**Value Iteration**

At gamma = 0.99 VI takes approximately 9.7 seconds and PI takes 1114.7 seconds.

Number of states has increased by 15 fold but time has increased at least 100 fold(check this number). The exponential increase is because for each additional state it must check all the neighboring states for each iteration. Thus adding more recursion.



*Figure : Time vs gamma, Large grid world*



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

**Policy Iteration**

Policy iteration can take a long time even though the equations for calculating the utilities are linear. When linear equations are being computed their matrixes are inverted. This process can still take a long time for large matrices. We have large matrices when we have a large number of states.

PI is usually faster because we do not need to find true utilities to find the policy. Why is that not working in this case?

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. If PI does not converge in this case it is because it is too computationally expensive.

Both VI and PI cannot reach the goal state with policies found with gamma < 0.98. The PI and VI are not looking into the future and the reward is not propagating enough towards the start state. This results in low utilities that do not create a policy pointed towards the goal state. Instead the policy is in random directions. 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.

**SARSA**

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.

