Learning Algorithms and Quake

David Ganzhorn, William de Beaumont
University of Rochester
March 19th, 2004

Abstract

We experimented with implementing various classic learning algorithms. We implemented value iteration and modified policy iteration, as well as a few passive reinforcement algorithms, direct utility estimation, temporal difference learning, and adaptive dynamic programming. We then analyzed their behavior. We lastly created an agent in Quake that was capable of navigating through the world, and learning the layout of the region, while mapping where it had been. The Quagent used Value Iteration to find the optimal policies for exploring, and thus was able to explore the world in a highly efficient manner. It progressed from general exploratory wandering to seeking out specific points in the world to explore, as it learned more about the region it was in. We also discussed and determined what future work we would like to do on this Quagent as well as learning algorithms in general.

Motivation

Our primary motivation for this project was to become more familiar with classic learning methods. However, we also wanted to see how much we could accomplish with such methods in a complex environment. Thus, we decided to extend our previous efforts with Quake agents (Quagents).

Methods

The five algorithms we implemented were value iteration, modified policy iteration, direct utility estimation, temporal difference learning, and adaptive dynamic programming.

Value Iteration determines the optimal utilities by looking at each square in a grid world, determining the optimal direction to move, and updating the utility of the current square given the expected utility of the best action. Value Iteration must run for many iterations in order to stabilize the fluctuating values. Modified Policy Iteration is somewhat similar, as it determines the optimal policy over a number of iterations. Modified Policy Iteration determines the utility of the current policy, then creates a better policy based off of those values, and repeats until the optimal policy is determined. Between the two iterating algorithms, we found that the value iterator in general ran slightly faster, but we believe that is because we were not familiar enough with Modified Policy Iteration to optimize it as much as we optimized our value iteration algorithm. On a high-end computer, we were able to do value iteration on a 100 by 100 grid with a gamma of 1 and an error of one millionth, on randomly generated worlds, in roughly a third of a second. Our Modified Policy Iteration implementation was quite close, but we found the Value Iteration implementation to be a bit more consistent, and the minor bug that the former had, which is why we ended up choosing that one for our quagent.

The Direct Utility Estimation program was the first of the algorithms that did not initially know the transition model. We implemented DUE, TD, and ADP as passive learning algorithms, and thus they determined the utility of each square in the world, according to a given policy, but it did not determine the optimal policy. The DUE algorithm works by simulating numerous runs through the grid-world, and then altering the utilities of the squares traversed based on the ultimate total reward. Although the algorithm is simple, it is not very efficient as it does not take the transitional model into account, and thus it does not update the utilities in a way that conforms to the constraints of the grid-world.

The Temporal Difference algorithm is similar to DUE, except that it updates the utilities in a way that is approximately consistent with the constraints. It uses an equation to update the previous square immediately, based on the rewards and utilities encountered in one move. Because the algorithm does not take the transitional model into account, occasionally the algorithm will dramatically adjust a square based on an unlikely, and highly undesirable outcome. However, this happens rarely, and reducing the impact of new experiences as experiences accumulate mitigates this problem. Because TD only updates the utilities of a single square at a time, it runs vastly faster than ADP, and because it updates the utilities more intelligently that DUE, it converges to reasonable estimates in far fewer trials.

The last algorithm we implemented was adaptive dynamic

programming. Unfortunately we were not able to get the program running within the time frame of the project. We will omit it during results

None of the passive learning algorithms were suited for our Quagent, as the exploratory rewards in the world would only be present for the first time the Quagent reached them, and thus the Quagent needed to be able to predict where the rewards were, rather than only knowing that a one-time reward was in a given location. Also, the passive learning algorithms would not generate new policies, and thus would not be useful for determining where our quagent should move.

For our Quagent, in order to simplify the quake environment to one that was feasible to deal with, we discretized the quake world into a 20 unit by 20 unit grid (in Quake world, 20 units appears is approximately a foot or two). This partitioned the first level of Quake into a 130 by 130 grid. We ran our Quagent in a corner of the world, the first room, in order to run trials more rapidly. However, the program is certainly capable of exploring the entire first level, although the intelligent path finding for the larger area will result in a substantial performance hit.

Results

Much of our testing for Value Iteration and Modified Policy Iteration was done on the example grid-world given by Russel and Norvig (1). The standard world is a four by three world, with a good terminal state, a bad terminal state, and a single obstacle. The reward at the good state is +1,

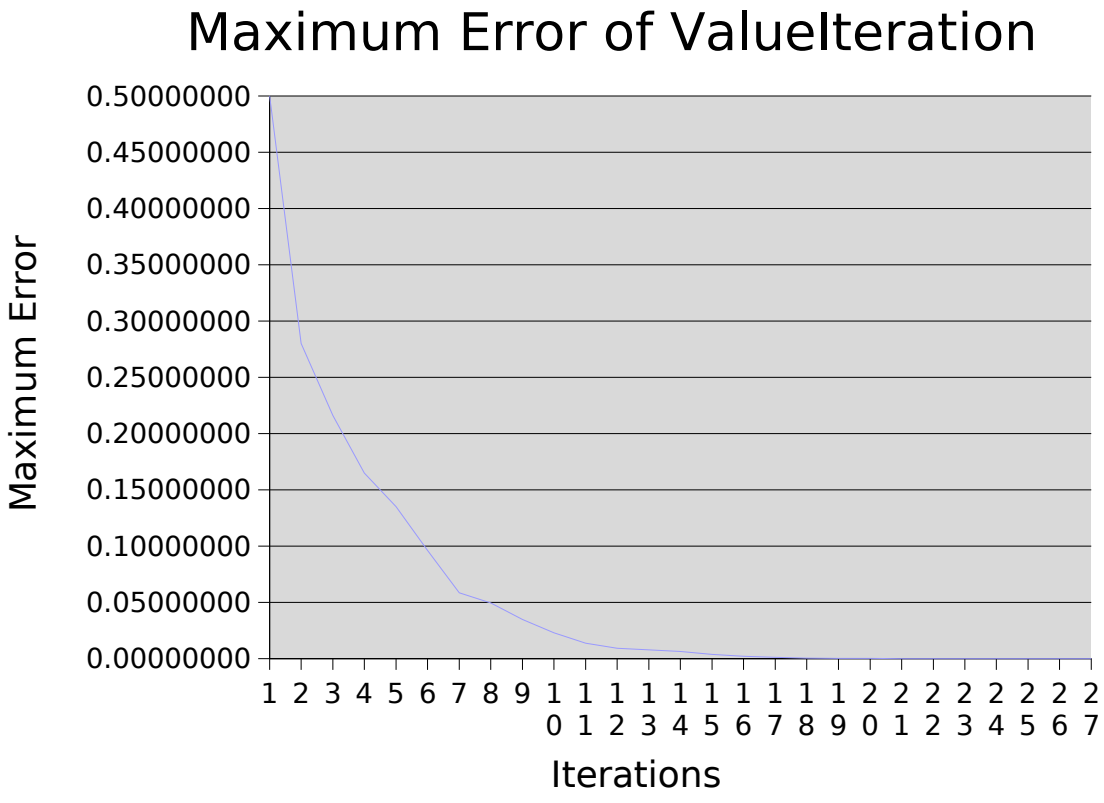
the penalty at the bad state is -1, and the cost of being in a non-terminal state is -0.04 per turn. Our Value Iteration program used the following representation of the grid world and from that determined the optimal policy.

Input	**Output**
***R	>>>R
*O*p	^O^p
****	^<<<

The specific utilities for this world were calculated to be:

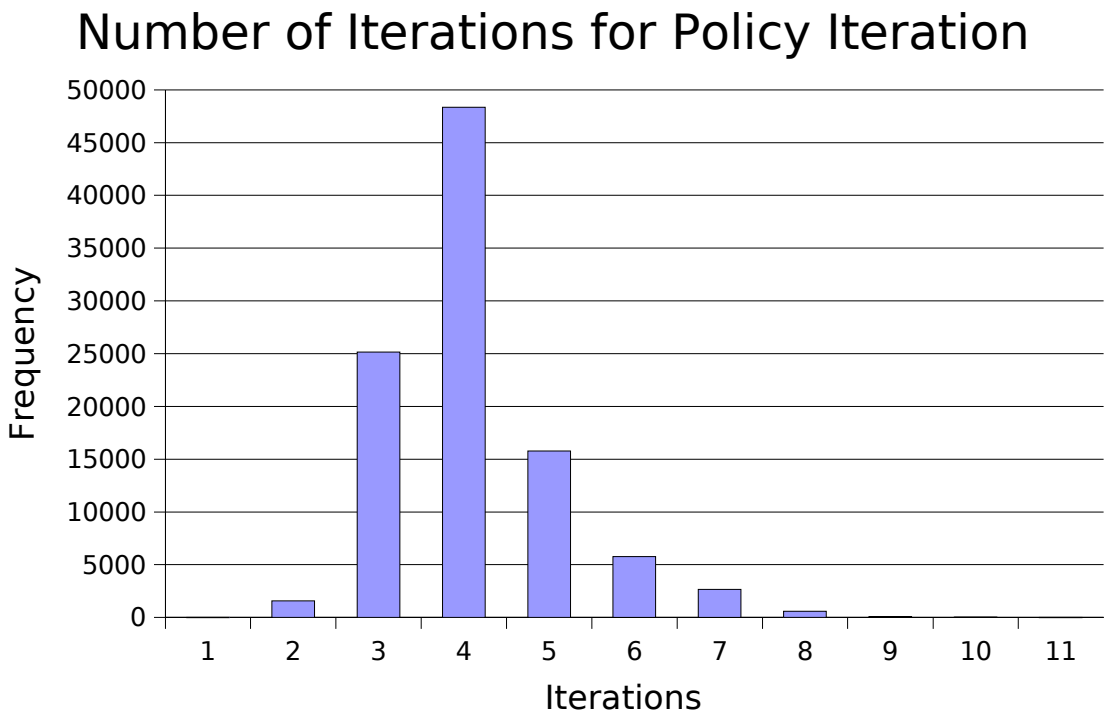
(0.812)	(0.868)	(0.918)	(1.0)
(0.761)	(0.0)	(0.66)	(-1.0)
(0.705)	(0.655)	(0.611)	(0.387)

Following is a graph of the maximum error of our Value Iteration program as it processes the standard world.



This instance of our Value Iteration program was set to converge on utility estimates until it was below an error threshold of 1/1,000,000. The algorithm converges to good estimates quickly, but takes a long time to converge to extremely accurate utility estimates.

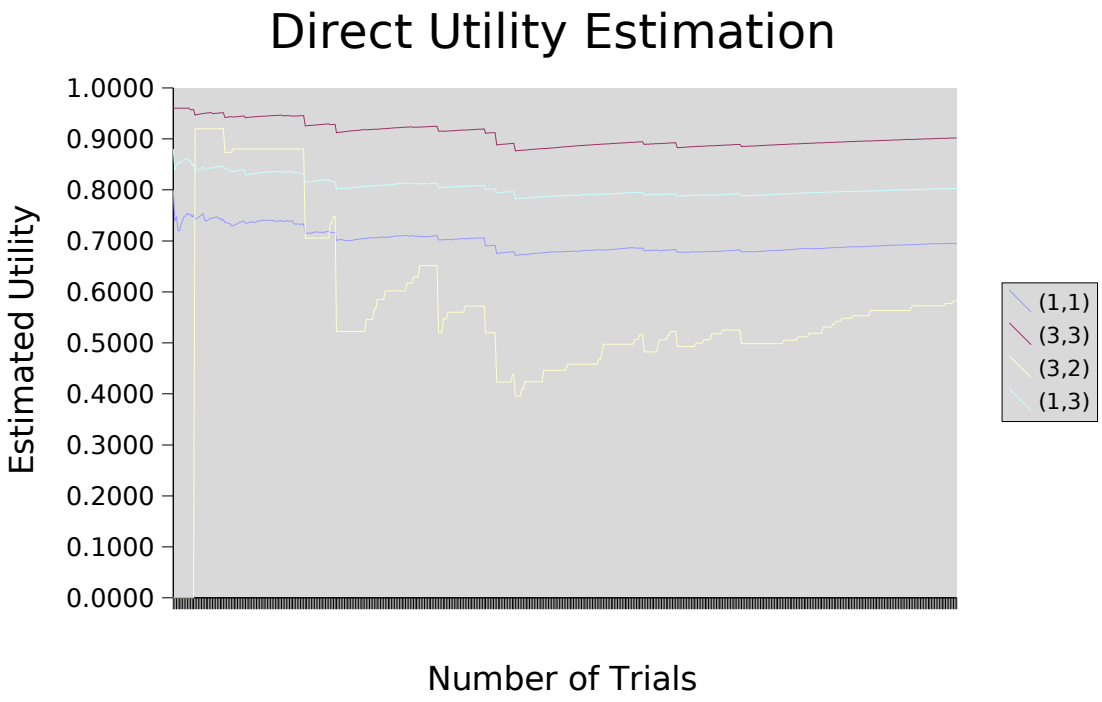
Our Modified Policy Iteration worked quite well in general, and often was much faster than value iteration, though sometimes it was slower. Following is a graph of the distribution of the number of iterations Modified Policy Iteration took to converge on the standard world, out of 100,000 trials.



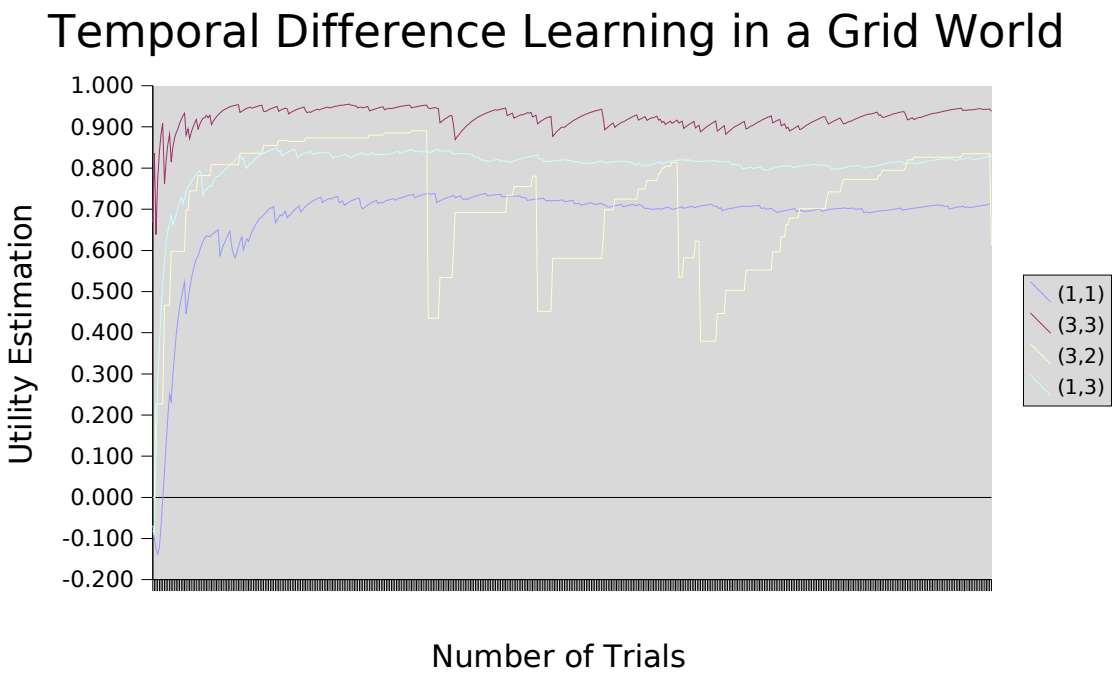
The Modified Policy Iteration algorithm always took far fewer iterations to converge than Value Iteration, but each if MPI's iterations were longer than the corresponding iterations for Value Iteration. For our utility estimation step in Modified Policy Iteration, we used up to 100 simplified

value iterations (simplified as the policy was fixed, and thus only one action had to be evaluated). Each iteration ended up taking several times longer than an iteration in Value Iteration.

Our passive reinforcement learning algorithms also had some interesting results. What follows is a graph of a selected set of utilities of individual squares, as computed by our Direct Utility Estimation program, over 500 trials. The algorithm works fairly well, but as shown by the data on the 3,2 square, the algorithm can take an enormous number of iterations to fully converge. This is because it does not make use of transitional information in order to learn that, because 3,2 is adjacent to 3,3, it should have a utility somewhat near 3,3's. However, direct utility estimation has a very low runtime per iteration, and it is possible to run millions of trials on a small world, in a single second. But, it would require far too many trials to accurately converge for much larger worlds.



Following is a graph similar to the preceding one, but for Temporal Difference learning. The graph shows 500 iterations throughout the algorithm. The enormous changes in the estimated utility in square 3,2 is due to the sudden swing in expected utility when the algorithm wanders into the penalty square a few times in a row, which is an unlikely event, and why there are only three or four major plunges out of 500 trials. The algorithm tended to find rough estimates very rapidly, but then was unable to converge to very accurate values. We considered using it in active form for navigation for our quagent, as we did not need perfect perfect behavior, but in the end we decided that it would still require far too many trials, even though it converged relatively quickly.



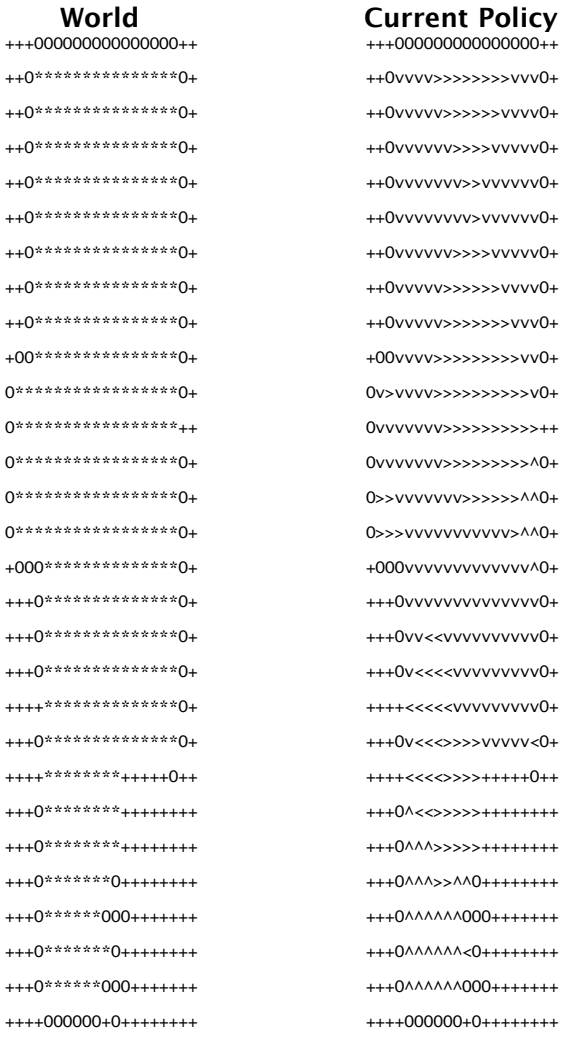
(Learning over 500 trials)

The Quagent we created was restricted to exploring the first room of the Quake level we tested it on, due to time constraints. The following

display is an example of output from when the quagent had nearly finished exploring the room. The O's are walls, the +'s are unexplored regions, the *'s are explored regions, and on the policy map, the arrows point in the optimal direction to move. The only rewards are unexplored regions, and all other squares give a small penalty.

The initial condition of the world is a completely unexplored grid; any direction the agent decides to go in is optimal. Because the Quagent's movement is stochastic (it has only an 80% chance of moving in the intended direction, and a 10% chance of veering to the left, and a 10% chance of veering to the right), the Quagent will initially move in straight lines, so that it doesn't risk backtracking. Once the Quagent has swept through most of the room, it will head towards the areas that are nearest and densest in unexplored regions, and eventually it will head specifically for the nearest unexplored points when the majority of the map has been explored and only individuals points in the room are still unknown. The agent is capable of marking anything it collides with as an obstacle, in order to avoid wasting time running into walls. Also, if the agent happens to become stuck on some of the inescapable points in Quake world (inescapable due to movement bugs in Quake), it will re-spawn and continue to explore the room, but will never return to the point at which it was stuck (as it will have been marked as being a wall, as will every adjacent square). One can see two such inescapable points on the output map, as they are obstacles that are surrounded by

obstacles on all sides.



Discussions

We were surprised at the efficiency with which our Quagent was able to explore and map the room, and that it never became stuck in a behavioral loop or did any other undesirable behavior. Also, we were especially surprised at the robustness of the Quagent's exploring behavior when during one trial we attempted to block the bot from walking down a ramp; however, once the Quagent had moved partially

down the ramp and found that it could thus reach many new unexplored regions, it persistently tried to move around the player avatar to get to the new regions, despite our frantic efforts to block it's path before it could move out of the first room. After a minute, we were able to herd it back into the first room, and the Quagent, having determined that the ramp was completely blocked, did not attempt to head down it again.

We would have liked to implement ADP as as active reinforcement learning algorithm, but unfortunately we did not have the time to. We would have liked to use such an algorithm to control a Quagent that was capable of dealing with much more interesting environments. In particular, implementing such an algorithm would allow the quagent to be completely ignorant of the transitional model, and to eventually learn whatever transitional model was actually in effect. Many more trials would be necessary to train such a Quagent, but the more robust learning would be a significant benefit, as the Quagent could then deal with situations where the transitional model is different each run, or when the transitional model varies based on location.

We also would have liked to implement a Quagent that was capable of learning complex transitional models, such as a world that involves stairs and platforms on top of other platforms, in order for the Quagent to be capable of dealing with all of the environments in Quake.

Overall, we are satisfied that we now have a better understanding of learning algorithms, and an exploring Quagent that behaves surprisingly efficiently.

Works Cited

(1) Stuart Russell and Peter Norvig. "Artificial Intelligence: A Modern Approach, Second Edition." New Jersey: Pearson Education, Inc. 2003