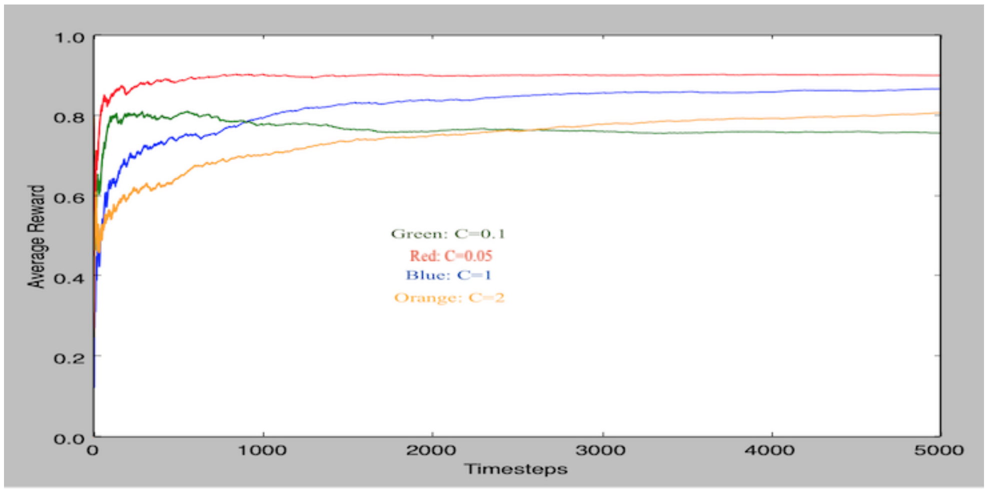
Assignment 1 - CPS 841 Reinforcement Learning

By: Daniel Bellissimo

Due: January 31st

1 UCB Algorithm Analysis

Through running the program we can observe how the agent slowly learns to take the best actions. Depending on the probability values for the arms and time step t, every 100 time steps the agent takes around 4-9% more optimal actions than the previous up to a certain limit. Overall after around 700-1000 time steps the agent has discovered enough about the environment to near reliably take the action which guarantees the highest possible award. I tested the agent using different random environments with various constant values for the action selection function. This constant value determines the amount of "confidence" assigned to levers which have been pulled few times. Through testing the algorithm, I found that very small constant values somewhere between 0.05 and 0.1 enable the agent to learn the fastest. Towards infinity however, this advantage is not visible as any constant value between 0.05 and 1 converges to the highest average reward value of 1.0 . With constant values between 0.05 and 0.1 after approximately 700 iterations the agent begins almost always taking the most optimal action.



In the image above we see an example of how a constant value of 0.05 learns the fastest in 5000 timesteps

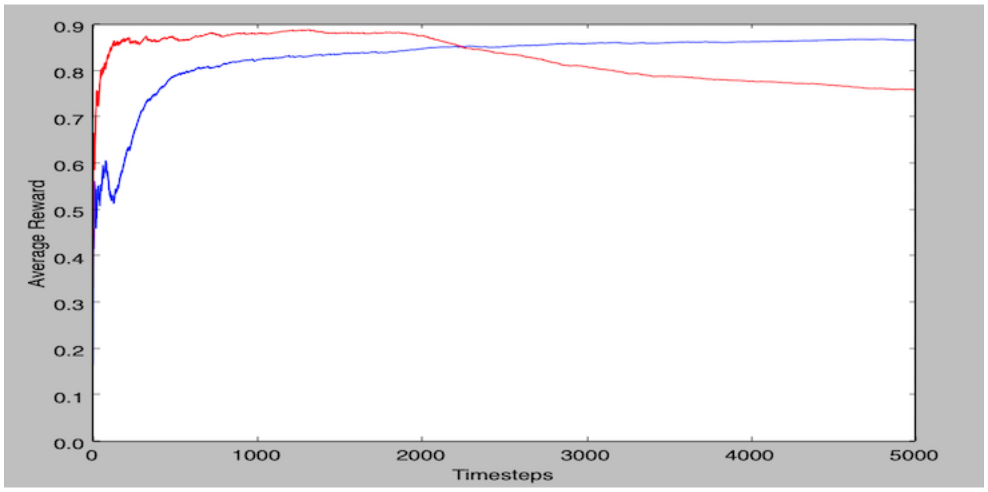
This indicates that in order to maximize learning speed with UCB, we must assign a low confidence level to levers not yet pulled. Also the environment does have an effect on the learning speed, since we naturally begin learning by starting with the first arm, depending on the constant value it may take some time steps for the algorithm to eventually work it's way to the other arms if the first arm has a good but not optimal reward probability.

2 Linear Reward Automata Analysis

Linear Reward Automata uses a probability distribution for the action selection, and updates the distribution at each time step in two ways depending on if the arm pull action was successful or not. In the succesful case, the probability distribution gets updated so that the arm that just had a successful pull will have an even greater probability of getting pulled in the next time step, while all other arms will have their respectful probabilities of being selected decreased proportionally. The inverse occurs when an arm pull is unsuccessful. This automata uses two constants, alpha and beta. The alpha constant scales the probability increase of successful arm pulls, while the beta constant scales the probability decrease of unsuccessful arm pulls.

2.1 Linear Reward-Inaction

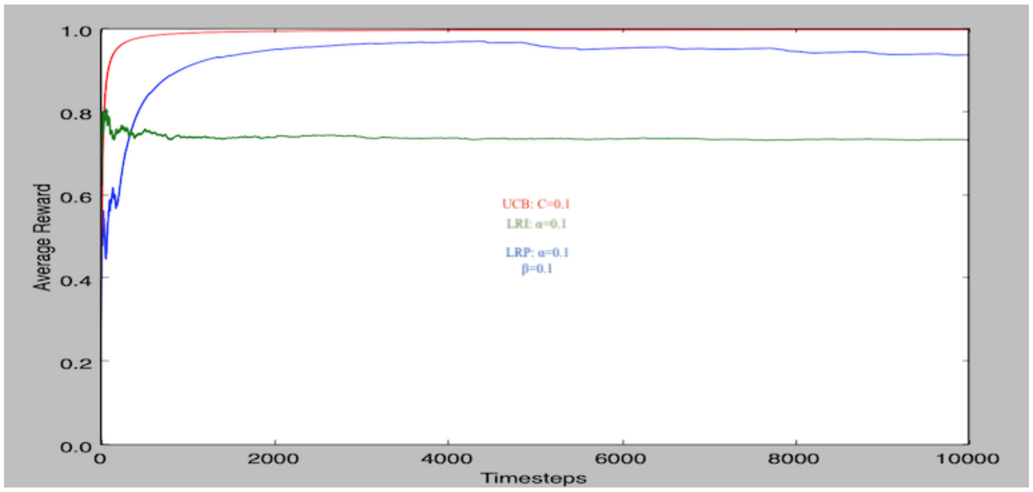
In Linear Reward-Inaction automata the action probability distribution only gets updated on successful actions and not unsuccessful actions this can cause the agent to have a predisposition to continuously pulling a arm it randomly selects if it gets a few reward signals from successive arm pulls. This causes a snowballing effect where the agent never really ends up exploring the other arms. The agent behaves very randomly using this learning automata, sometimes it finds the most optimal action and continuously pulls it, othertimes it never finds the optimal action arm and instead finds a less-optimal arm. This means that two linear reward-inaction tests on the same environment will always have different results, sometimes those results differ wildly. Take for example the following graph of two Linear Reward-Inaction agents using a alpha value of 0.1 and the same environment.



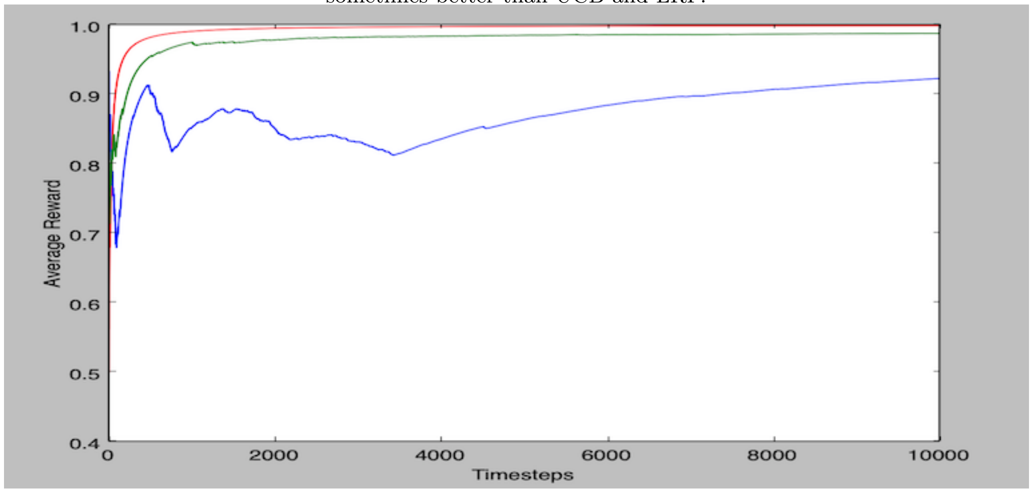
In the image above we see the performance differences between two LRI automata working on the same environment

2.2 Linear Reward Automata VS UCB

When comparing both Linear Reward automata with UCB we notice that UCB consistently performs the best against both LRI and LRP. In the following image, notice how consistently the hyperbolic curve of the UCB algorithm approaches the upper reward limit. Linear reward-penalty performs slightly worse compared to UCB, while Linear Reward-Inaction's performance depends more on the environment and how it randomly selects which arm to pull. This means that in some cases Linear Reward-Inaction function is bounded by a function of UCB and LRP's performance, except for very few cases where both Linear Reward automata are better than UCB, which is when the first arm pulled is the best arm, because UCB still explores the other arms while Linear Reward just keeps pulling. Meaning, sometimes LRI can perform better than LRP, and sometimes both of them perform better than UCB. Since UCB is much harder to implement in more general RL settings, LRI might be a better choice between the three but it can be random, most of the time however, LRI is better than LRP.



Above graph is comparing UCBvsLRIvsLRP, while the graph below shows how the LRI algorithm can also perform nearly as and sometimes better than UCB and LRP.


Average Reward
Timesteps