Exploration versus Exploitation Dilemma in Reinforcement Learning

Michael Piñol

California State University Stanislaus

CS4960

Dr. Melanie Martin

Abstract

The dilemma of exploration versus exploitation in reinforcement learning problems has no single solution. There exist many algorithms whose approaches can achieve near optimal rewards however each of these approaches is vastly different from one another. The E-greedy algorithm reduces the issue of exploration to a simple probability where E3 explicitly explores by exploring when a state is unknown and when a non-optimal action exists in a particular state. R-Max's approach is to implicitly explore all states by assuming every action returns an optimal reward until it learns otherwise and REX, which is the most sophisticated, attempts to use an informed exploration model where objects and states are generalized so that even in previously unencountered states, with previously unperformed actions, an informed decision can be made as whether to explore or exploit, on the basis of its knowledge of previously known states and performed actions. Each of these algorithms demonstrates a different approach to the exploration versus exploitation dilemma where all perform near-optimally.

1. Introduction

Reinforcement learning can be described as learning what to do in an unknown environment in order to maximize some numerical reward. Reinforcement learning problems typically consist of some learning agent, existing in an unknown environment that attempts to learn about its surroundings by performing various interactions with it. Each action performed by the agent in the environment grants the agent some numerical reward while also causing some aspect of the environment to change. The agent's ultimate goal is to learn, through its various interactions, a way to achieve as maximum as possible a total numerical reward by performing actions within that environment. (Sutton & Barto, p. 2)

At the initialization time of reinforcement learning problems the environment is unknown to the agent resulting in an agent's need to first learn about its environment by performing unknown actions, known as exploring. Performing actions is how the agent learns about the environment, its changes, and also the corresponding reward values for performing particular actions. After the agent has run for a particular amount of time it will have gained some approximate knowledge of the environment and the various rewards attributed to some actions which allow for the agent to begin actively attempting achieving a maximum as possible total reward. At the time an agent can begin to effectively exploit its knowledge there may still exist actions that have not been performed and therefore actions exist whose reward value is still unknown. This means that there is some possibility of an existing action, unknown to the

agent, whose reward is greater than the currently known rewards for that particular state meaning the agent is receiving non optimal rewards.

The goal of the algorithms that will be examined is to achieve as near optimal a total reward as possible, an optimal total reward being the result of an agent performing only the most optimal actions that achieve the greatest possible maximum reward possible for a particular stochastic game, and acting in an unknown environment it is highly improbable that an agent will be able to collect an optimal reward without first becoming knowledgeable about its environment. This raises the dilemma of how long should an agent spend exploring its environment versus exploiting its knowledge. (Sutton & Barto, p. 2) At which point can an agent be considered knowledgeable enough about its environment to exploit its knowledge and how much exploration is required for this state to be reached? This paper will examine the E-greedy, E3, R-Max, and REX reinforcement learning algorithms and their approaches at addressing this dilemma.

2. Reinforcement Learning Problem

A typical reinforcement learning problem will consist of some stochastic game and a learning agent. The learning agent plays, operates within, the stochastic game for a particular number of times in an attempt to perform more efficiently, optimally, with each subsequent game. (Sutton & Barto, p.3)

2.1 Stochastic Modeling and Games

Stochastic modeling is the "use of probability to model real-world situations in which uncertainty is present." (Glynn, p. 1) Stochastic modeling is a tool used to reduce the many aspects of various situations to probabilities of possible outcomes. This way of modeling is a great tool for reinforcement learning as it allows for aspects of the real world to be accurately represented in a meaningful way so that learning algorithms may then be applied.

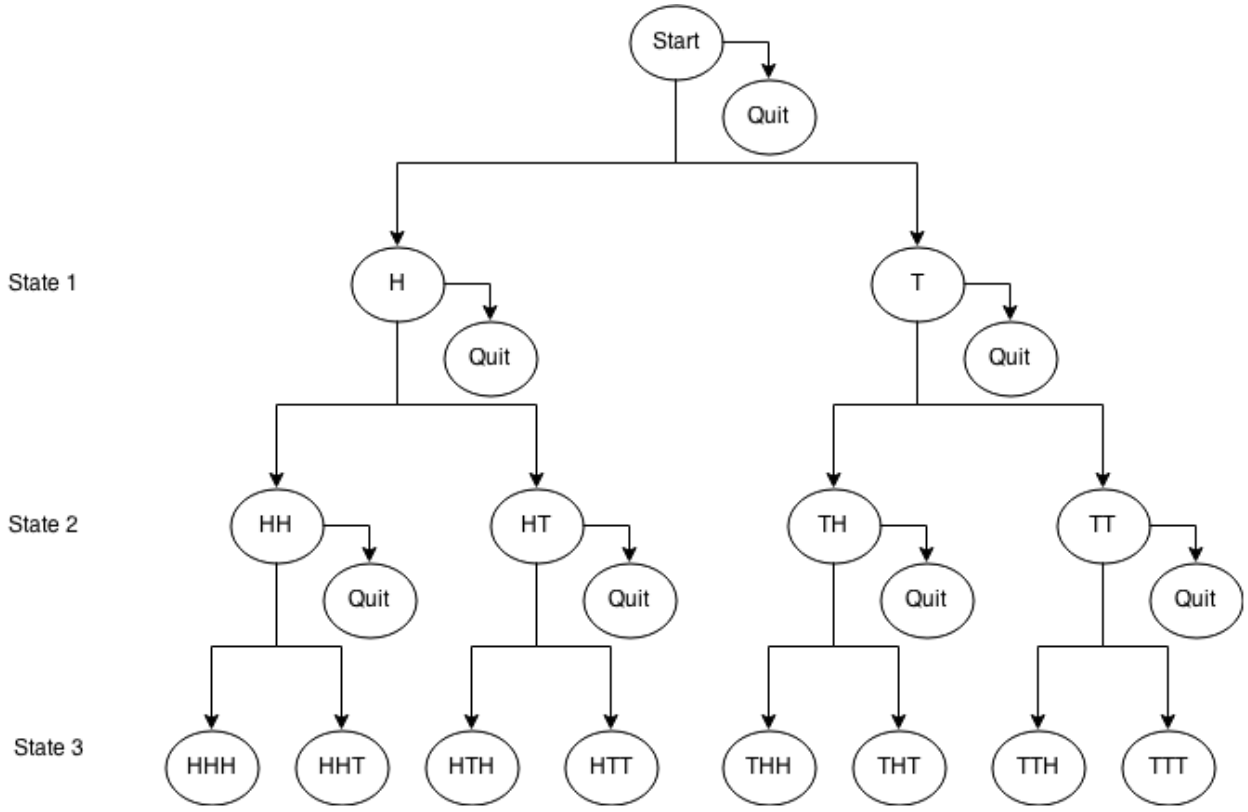

Figure 1 Represents a simple coin toss stochastic game where there are two possible actions that can be performed from any particular state, either to quit the game and cause no more state transitions or flip the coin. H and T represent the outcome of a flip action where H is heads and T is tails. The nodes represent the current state, which depict the outcomes of all previous actions prior to arriving at that state. The connecting edges from a state represent an action taken, whether to flip the coin or to stop the game. NOTE: when the quit action is performed the game does not reach the successive state.

A stochastic game is a simulation of a stochastic model and is commonly comprised of a set of states, state spaces, and some probability attributed to each action within the state space to cause a state transition. (Glynn, p. 2)

As an example, a state in the stochastic game Solitaire would be the current configuration of all of the cards on the playing area. The state space would then be every possible move, or action, a player could make in that particular state. It is important to note that as the game is played the state is expected to change after every move and as a result the set of all possible moves also changes. (Givan & Parr) This is one of the basic aspects of a stochastic game and why the exploration versus exploitation dilemma exists in reinforcement learning. Games with a great number of states, each with a great many actions, make it difficult for reinforcement learning algorithms to meaningfully learn about the possible states and explore the even greater amounts of possible actions. A comparison of the simple stochastic game of flipping coins to solitaire can be employed to demonstrate this concept.

Think of a simple coin flipping game where an unbiased coin, where one side is heads and the other tails, is flipped at most three times and the player can choose to stop playing at any time. At the end of the game if the number of heads is greater than the number of tails, the player is given that amount of some reward, if tails is greater than the number of heads, the player loses that amount of some reward, and in the case of one head and one tails being flipped with the player deciding to stop the game, a reward of one is given. This simple coin toss game, represented by a tree in Figure 1, has a possible 14 states, 8 of which conclude the game, where only 6 provide the player with a state space with the possible actions of either flipping the coin

or stopping the game. Since the number of states is so few, even an inefficient implementation of a learning algorithm playing this game would be able to act optimally after only a few iterations of playing the game. (Gimbert, p.13)

The game solitaire however has a vast amount of states, as well as state spaces, as the possible configurations of the deck of 52 cards across the many zones on the playing area are exceedingly large. This game would require a lot more time for a learning algorithm to be able to act optimally in any one particular state.

2.2 Agent

An agent can be thought of as an abstract individual learning entity. The typical components of an agent consist of an environmental model, value function, reward function, and a policy. The implementation of an agent does not necessitate that all the listed components be present, even amongst the algorithms being examined this is not the case, just that this is a typical implementation. In most cases the agent can be thought of as the actual 'player' of the game. (Sutton & Barto, p.7)

2.2.1 Model of Environment

The model of environment is the agent's representation of its current working environment. This model is referenced by other functions to determine whether a state has previously been

visited, whether all actions have been explored, and whether an optimal action exists in this state. In the game of Solitaire an environmental model might be implemented to record each configuration of the playing area, the move made, and the new configuration of the playing area as a result of the move made. (Sutton & Barto, p. 8)

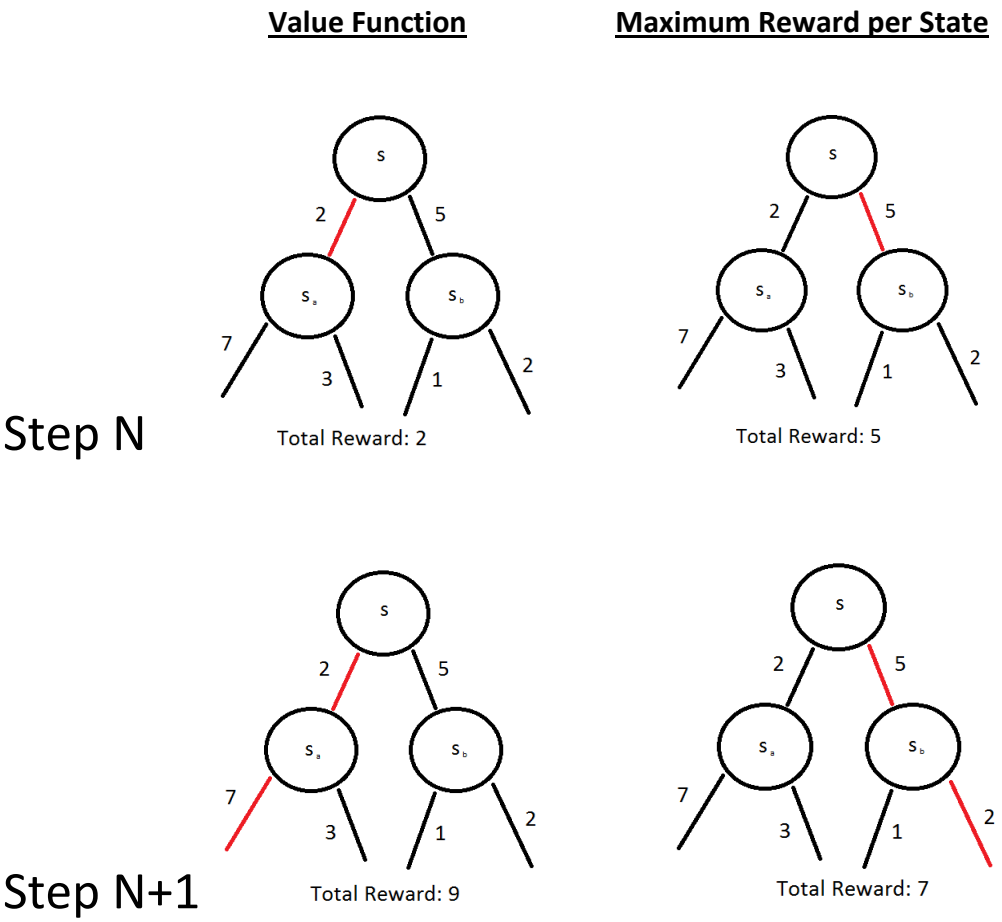

Figure 2 A simple stochastic model which has three states, {S, Sa, Sb}, and where each edge from those states represents a particular action with its corresponding reward value.The edges marked red represent a performed action at a particular step in time. The left two graphs demonstrate how an agent's value function might create a plan so as to recieve a higher reward across multiple states where the right two graphs demonstrate how an agent utlizing a local maximum reward method of obtaining rewards would behave on the same graph.

2.2.2 Value Function

The value function utilizes the environment model to derive long term plans so as to maximize the agent's total reward. To clarify, the value function compares the rewards across possible future states by using the environmental model to determine the probability that any one particular state might follow a particular action. This allows for the agent to predict future states and forego current state maximum rewards for the possible greater total action in the future. Since future state predictions rely heavily on the environmental model the environment must first be sufficiently known by the agent for a plan to emerge that attempts at achieving a greater total reward. (Sutton & Barto, p. 8)

Assume an agent has played a simple stochastic game; represented in figure 2, sufficient enough times so that its model of the environment is very close to be accurately representative of the real working environment. Each of the three states, {S, Sa, Sb}, has two possible actions that can be performed by the agent from that state. Each action in state S leads to either state Sa or Sb and regardless of the action performed in either state Sa or Sb the subsequent state reverts to S. If an agent were to operate under the policy of obtaining the maximum reward from any particular state at step N it will perform the action that rewards a value of 5, causing the state to change Sb, where at step N+1 the action that rewards a value of 2 will be performed for a total reward value of 7.An agent's value function affords the agent the ability to plan ahead and create action sequences across multiple states. Operating in the same game with sufficient knowledge of the three states and state spaces, an agent with a planning ahead value function, at step N would perform the action with a reward value of 2, causing the state

to change to Sa, where at step N+1 the action that rewards a value of 7 will be performed for a total reward value of 9.

In the case of solitaire, a value function operating over the environment model might allow for the emergence of a simple strategy like forgoing a card placement in order to reveal a hidden card or in the case of the coin toss game, stopping the game at step 2 because the agent has received two heads which provides a reward of value 2 instead of flipping one more time with a 50% probability of getting a head at step 3. The agent forgoes the action that might reward a greater or lesser reward for a guaranteed positive reward.

2.2.3 Reward Function

An agent's reward function determines the perception of a received reward. More specifically, it determines whether a reward was 'good' or 'bad' by comparing it other rewards received from the same state and also to rewards received from other states. In the simple coin flip game, heads would be perceived by the reward function as the return of a ‘good’ reward and that tails would be perceived as a ‘bad’ reward. However, the reward function might weight the return of tails depending on the state it appeared. A tails returned at step one might return a ‘worse’ reward than a tails received at step two after receiving a heads at step one because the probability of receiving an optimal reward is greatly diminished. The reward function also determines what is considered an optimal reward for the value function by improving estimates of non-optimal actions in the environment model as information is collected by exploration.

Essentially an agent's reward function "defines the goal" of the particular reinforcement learning problem. (Sutton & Barto, p.7)

2.2.4 Policy

An agent's policy determines its behaviour at any particular time and is based on results given from the value and reward functions. (Sutton & Barto, p. 8) The relationship between the value and reward functions and performed actions is similar to the idea of stimulus and response. That is, a state provides an agent some stimulus and the policy derives a response based on values returned by the reward and value functions. The example of how the reward function perceives the reward of tails depending on the state at which it was received exemplifies exactly this; as the stimulus of the new state will suggest that a particular action take place.

3. E-Greedy Algorithm

The E-greedy approach is a very early learning example algorithm whose solution to the exploration versus exploitation dilemma was to reduce the issue of exploration to a probability E. That is, the agent will have an E probability of exploring at any particular time step. This means that an E value of 0 will result in an agent that never explores its environment but only

exploits its current knowledge and an E value of 1 will result in an agent that never exploits its knowledge but only explores its environment.

Since E-greedy uses a probability model to determine exploration, it lacks a value function and cannot attempt to plan, it operates solely on the reward functions perception of state maximums derived from the environmental model.

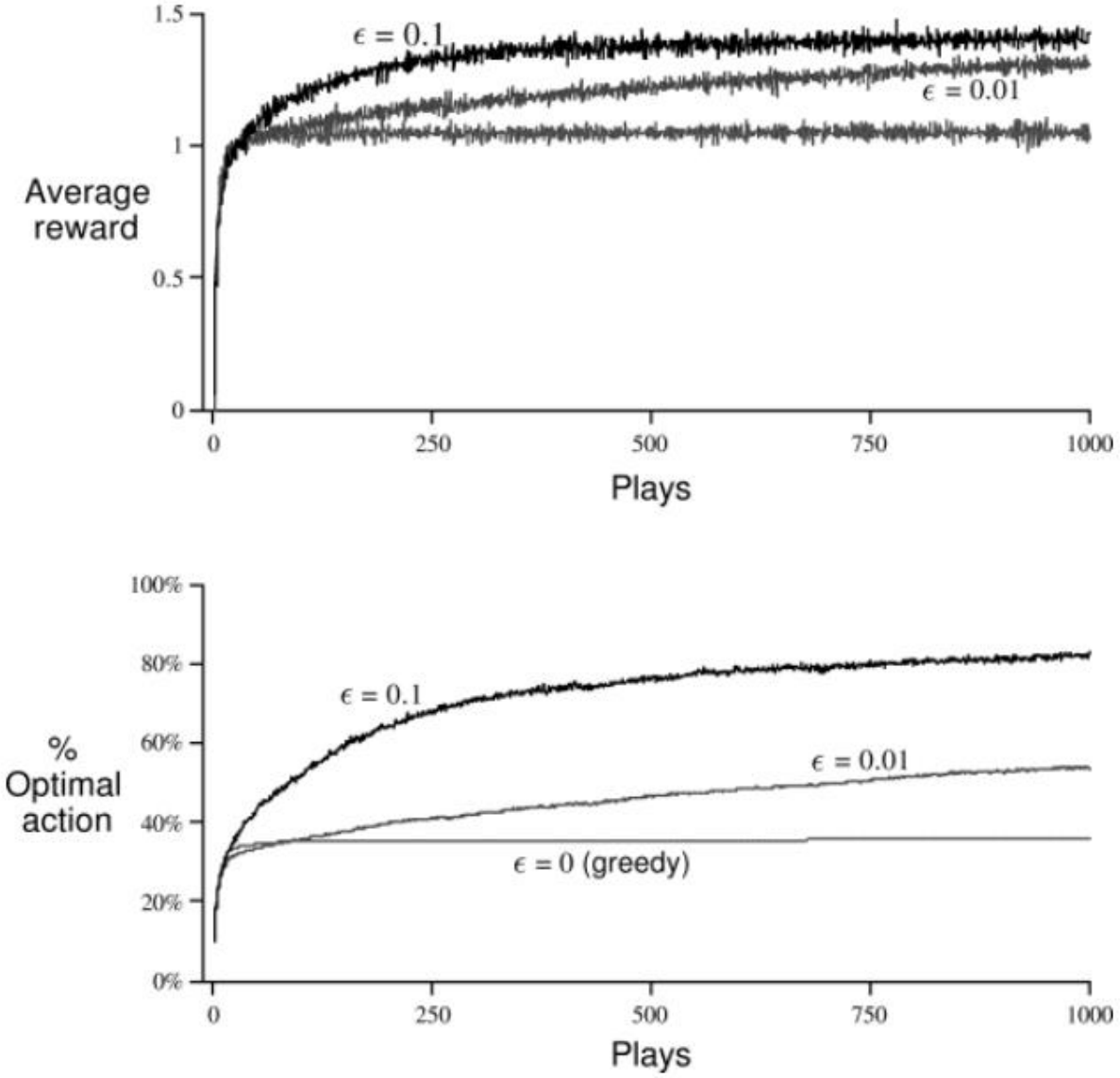

Figure 3 Performance of three E-greedy algorithms with different values for E. The tests were performed on a simple repeated game with a single static state and a state space of 10 for 1000 iterations. Each possible action performed by E-greedy supplied the algorithm with some numerical reward where the values were attributed to a particular action at random and remained constant throughout the entirety of the game. (Sutton & Barto, p. 29)

The average reward graph in figure 3 shows that after 1000 steps the E=0.1 algorithm achieved a much greater average reward and also that it achieved a greater average reward in fewer steps than both the E=0.01 and E=0 algorithms. The optimal action percentage graph supports this conclusion as well however it also represents the data in a manner where an important

inference can be made. The optimal action percentage graph shows that E=0.1 performed optimally 80% of the time and then appears to level off. It also shows that E=0.01 performed optimally 60% of the time but displays a steady rate of increase. If both of the algorithms were allowed to run for a greater amount of time it can be surmised, just by the nature of the values of E, that E=0.1 will ultimately achieve an optimal action percentage of 90% whereas E=0.01 will achieve an optimal action percentage of 99%.

This idea can be reduced to out of every ten actions performed by E=0.1, regardless of the whether the optimal action is known or not, one of the actions performed will be non-optimal. E=0.01 operates under similar circumstances but will perform a non-optimal action one out of every one hundred actions. This demonstrates a correlation between the amount of time an agent will be allowed to learn its environment and the achievement of near optimal performance. Given sufficient time E=0.01 would obtain a greater average reward than E=0.1 even though it would take much more time steps for it to do so. Finding an optimal value of E for a particular reinforcement learning problem depends heavily on the amount of time that an agent has to play that particular stochastic game. (Sutton & Barto, p. 26-31)

4. E3 Algorithm (Explicit Exploration)

The E3 algorithm's approach to solving the exploration versus exploitation dilemmas is to use an explicit exploration model that uses direct and planned exploration. That is, an agent will directly explore, by performing the action it has tried in that state the fewest amount of times

in that particular state, until a state is known well enough where the greatest rewards for that state can be approximated causing planned exploration, when E3 encounters a known state it first attempts to find a near optimal action to exploit based on values returned to it by the value function and if an optimal action is not found the most well-known state action pairs are set to return a reward of zero and the remaining actions are set to return an optimal value temporarily in the reward function, to occur. Sutton & Barto, (p. 39-40), describe this as "optimism in the face of uncertainty" and this approach attempts to cause the agent to encounter previously unknown states by performing series of non-optimal actions. (Kearns & Singh, 2002)

A state will only be considered known if the visitation count value of that state reaches a threshold derived from the environmental model, which is represented as the record of the visit counts for each state as well as the actions performed from that state as a state action pair K(s, a), as the number of visit counts, the size of the state space, and the number of actions performed from that state. Essentially the threshold is determined by the number of times the state has been visited to the amount of actions performed in that state, to how many total actions exist in the state space. E3's threshold value relies upon the pigeon-hole principle that after N iterations, where N equals the number of possible states, states can begin to become approximately known. A state with a relatively small state space becomes known much quicker than a state with a large state space. (Kearns & Singh, 2002)

5. R-Max Algorithm (Implicit Exploration)

The R-Max algorithm approach uses an implicit exploration model which utilizes the idea of optimistic initial values. (Sutton & Barto, p. 39)That is, as an initial condition the agent will assume that every possible action in every state provides the optimal reward. This implicit approach requires the agent begin with some preconceived model of the environment that is updated as it encounters various state changes by performing actions. It is not required that this preconceived model need accurately represent the environment; it only need be some approximation that the agent can adapt. The environment model is updated in very much the same manner as E3 by recording all state action pairs, K(s, a). (Kearns & Koller, 1998) There exist however the distinction that since R-Max was designed to be used in multiplayer stochastic games and E3 was designed to operate on single player stochastic games, R-Max instead records joint state action pairs, where joint state action pairs are combinations of sequential state action pairs amongst actions performed by all players. (Brafman & Tennenholtz, 2002)

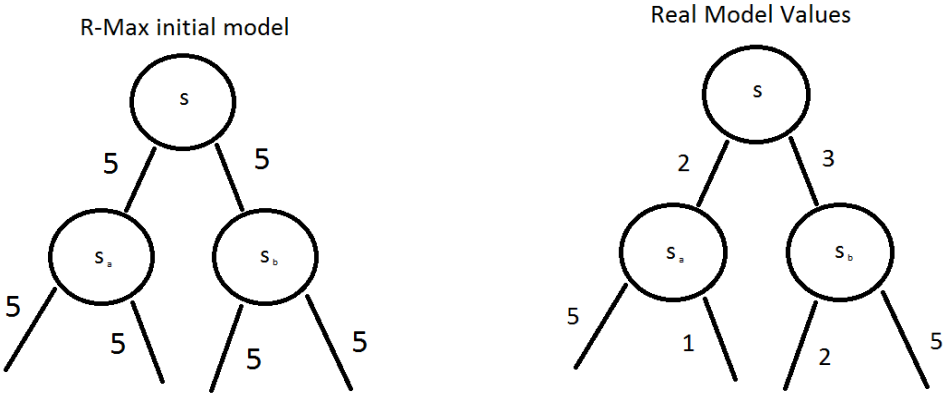

Figure 4 Representation of a simple three state stochastic game. The left graph displays R-Max's initial model view of the environment. The right graph displays the values of the actual environment. Each connecting edge represents a possible action from that state with its corresponding reward.

Inherently this algorithm spends much of its time exploring at the beginning of run time so that the vast majority of the total running time later on is spent exploiting. This also allows for the unique situation where after the agent has learned an optimal action for a particular state, there is still a possibility that it will unknowing exploit that action early on. That is, if the current state space has one known action that rewards an optimal reward and another unknown action that the agent believes rewards an optimal reward, there is a 50% chance that the agent will perform the optimal action at that particular time step. By simple probabilities, if there exist actions that reward optimal values, R-Max has the slight chance to perform optimally in very early stages of running that overall will produce a greater total reward.

6. Expanded Algorithm (REX)

The REX algorithm's approach uses an informed exploration model which utilizes objet generalization so that when exploration occurs, the agent can apply results from that action across actions in other states, increasing known state action pairs, and potentially reducing the amount of state action pairs that need to be explored. This is a great improvement over the previously examined algorithms as this potentially greatly reduces the amount of time required for states to become known meaning that exploitation can occur much earlier. (Lang &

Toussaint & Kersting, 2012) Object generalization can be best described by relating the game Pac-Man.

Both the E3 and R-Max algorithms must experience every action in a state in order to create a model that allows them to act optimally. (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002) This means that if these algorithms were applied to a game of Pac-Man the player must have their turn ended by each of the four ghosts in order to learn to avoid all ghosts. With REX's object generalization if a turn has been ended by a single ghost it can apply the learned behaviour over all ghost as the simple rule, avoid ghosts. This allows for the compression of state action pairs across all pairs, which not only reduces the number of times an agent must explore, but also potentially greatly reduces the amount of unknown states every time a new unknown state is encountered. It is important to note that state and state action pair compression operates on the assumptions that all states' features remain constant unless directly changed by the agent and that the agent is naturally uncertain about its definitions of objects. This allows the agent to update its generalizations over objects as new information is perceived over time.

While object generalization works to compress the total number of states and actions, it is the transitional model, the probability that an action will result in a particular state, and model of environment that indirectly determine whether to explore or exploit at a particular time step. This is accomplished by both models estimating a possible visitation count for a particular state using the compressed data and depending on whether the returned value reaches a particular threshold, not unlike the E3 algorithm, either the E3 or R-Max algorithm is run over the current

state space. REX attempts to utilize all of the positive features of both E3 and R-Max by determining which algorithm is suitable for a particular situation. This is how the idea of informed exploration emerges. When REX becomes knowledgeable enough of the environment it is able to make informed decisions on which algorithm would best suit the current state. (Lang & Toussaint & Kersting, 2012)

7. Conclusion

The exploration versus exploitation dilemma is difficult problem to approach as in many cases it is far easier to tailor an algorithm to suit the needs of a particular stochastic game than creating one that is all encompassing. This dilemma is made even more difficult by the relationship between the number of possible states and actions for a particular stochastic game greatly increasing the amount of time an agent will require becoming sufficiently knowledgeable about its environment. The E-greedy algorithm only explores by a probability of the value E and E3 and R-Max must explore every possible action to sufficiently know a particular state. If the agent exists in a sufficiently complex stochastic game with a large number of states and state spaces, then its ability to efficiently learn about its environment is dramatically reduced. REX provided a method to address this issue by its use of object generalization however, at its core it still utilized the E3 and R-Max algorithms.

The exploration versus exploitation dilemmas is a still ongoing research problem and has a great many different approaches not examined in this paper. It has still yet to achieve a

standardized approach that works across all reinforcement learning problems and based on the possible complexities of some of the problems that exist, there may never be a single set standard for exploration versus exploitation dilemma. Instead it might be more appropriate for a standard collection of varying approaches to exist where each one handles a particular set of problems.

8. Possible Future Work

It would be worthwhile to compare the effectiveness of each of these algorithms against one another to determine which performs more optimally and under which circumstances. I propose a suite of stochastic games be played by each of the algorithms compared. All games should be as different as possible in number of states as well as average state spaces. A single game from the suite should be played by the agents at least three different times where the amount of time allotted to the agent is increased so as to obtain information how state size, state space, and learning time affect learning.

9. References

Sutton, R., & Barto, A. (1998).*Reinforcement learning an introduction*. Cambridge, Mass.: MIT Press.

Gimbert, H. (2011). *Stochastic Games*[PDF document].Retrieved March 6, 2015, from Laboratoired' Informatique Algorithmique: Fondementset Applications web site: *www.liafa.jussieu.fr/~zielonka/Jeux/.../GT%20hugo%20v2.pdf*

Lang, T., Toussaint, M., & Kersting, K. (2012). Exploration in Relational Domains for Model-based Reinforcement Learning. *Journal of Machine Learning Research,13*. Retrieved March 5, 2015, from http://www.jmlr.org/papers/volume13/lang12a/lang12a.pdf

Brafman, R., & Tennenholtz, M. (2002). R-max – A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research,3*. Retrieved March 5, 2015, from ttp://www.jmlr.org/papers/volume3/brafman02a/brafman02a.pdf

Kearns, M., & Singh, S. (2002).Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning,49*(2), 209-232. Retrieved March 7, 2015, from http://web.eecs.umich.edu/~baveja/Papers/MLjournal6.pdf

Kearns, M., & Koller, D. (1999). Efficient Reinforcement Learning in Factored MDPs. *IJCAI'99 Proceedings of the 16th International Joint Conference on Artificial Intelligence,2*, 740-747.

Givan, B., & Parr, R. *An Introduction to Markov Decision Processes*[PDF document].Retrieved March 10, 2015, from Rice University web site: http://www.cs.rice.edu/~vardi/dag01/givan1.pdf

Glynn, P. *Introduction to Stochastic Modeling* [PDF document]. Retrieved March 12, 2015, from Standford University web site: w*eb.stanford.edu/class/cme308/OldWebsite/notes/ProbReview.pdf*