Spring 2018

Multi-objective Path Finding Using Reinforcement Learning

Prashant Thombre
San Jose State University

Multi-objective Path Finding Using Reinforcement Learning

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Prashant Thombre

May 2018

The Designated Project Committee Approves the Project Titled

Multi-objective Path Finding Using Reinforcement Learning

by

Prashant Thombre

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2018

Prof. Katerina Potika	Department of Computer Science
Prof. Mark Stamp	Department of Computer Science
Prof. Robert Chun	Department of Computer Science

ABSTRACT

Multi-objective Path Finding Using Reinforcement Learning

by Prashant Thombre

Path Finding is a vastly studied subject in the field of Computer Science. The problem of path-finding is defined as the discovery and plotting of an optimal route between two points on a plane. The existing algorithms that solve this problem are mostly static and rely heavily on the prior knowledge of the environment. They also require the environment to be deterministic. However, in real-world applications of the path-finding problem, often the environment is priorly unknown and stochastic, and with several conflicting objectives. In such cases, the aforementioned algorithms fail to produce effective results. In this project, we study and use a reinforcement learning approach for solving the many-objective path-finding problem, called Voting Q-Learning (VoQL), a model-free, on-policy learning algorithm. In this project, a set of optimal policies is determined with the help of the VoQL algorithm. This algorithm uses various voting methods borrowed from the field of social choice theory for action-selection. In addition to working with the existing methods for VOQL, the performance of additional voting methods is studied and evaluated for the first time.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude towards my graduate project advisor, Prof. Katerina Potika, for her valuable guidance and feedback throughout this graduate project. I would also like to take this opportunity to thank my committee members, Prof. Robert Chun and Prof. Mark Stamp, for their time and efforts. I also appreciate the feedback from my peers regarding my work on this project. Finally, I would also like to thank my family members and relatives for their constant support and motivation throughout the degree program that helped me keep up the spirit and complete the research work.

TABLE OF CONTENTS

CHAPTER

LIST OF FIGURES

CHAPTER 1

Introduction

Path finding is a very well studied problem in Computer Science field and it has numerous real-world applications, such as determining the shortest network route, autonomous robot navigation, detection of shortest path between source and destination on a map, etc. If we consider the environment to be a graph, then, at its very core, all the path finding problems address the question of how to reach a destination node from a starting node in a graph. This can be done by implementing a graph search algorithm for the given problem, which searches the graph starting at an arbitrary node and exploring the adjacent vertices of the visited nodes until it reaches the destination node.

The problem of finding any path between two nodes in a graph is just one of the two primary questions that path finding tries to answer. The other question that can be answered by a path finding problem is that of determining the most optimal route between the start and the destination node. Most frequently, we are concerned with the later problem, where we find an optimal path that avoids obstructions and minimizes the deviations due to those obstructions from the optimal path as much as possible before reaching the destination. Since, this is an optimization problem it is more complex and difficult to solve as compared to finding any path between the two nodes.

Different strategies are applied to solve the optimal path finding problem such as using greedy technique in Dijkstra's algorithm, dynamic programming technique in Bellman-Ford algorithm, use of heuristics in A* search algorithm, etc.

In most cases, algorithms for path finding problems have an inherent assumption that there is only one objective that the software program is trying to achieve such as, minimizing the overall distance traveled or reducing the time taken to travel from one point to the other. It is also considered that the environment will be deterministic and fully known in advance. However, in real-world applications we frequently work with initially unknown and stochastic environments. Also, it is more likely that instead of a single objective, the problem at hand needs to consider multiple conflicting objectives.

The validity of the solution provided by above mentioned algorithms becomes void when the environment changes. Due to the non-deterministic nature of the environment and more than one objectives, the static algorithms mentioned above decrease in effectiveness or become completely ineffective. So, it is necessary to use a method that could obtain efficient paths in an unknown and stochastic environment where a sequence of decisions need to be made to reach a set of objectives. Reinforcement learning (RL) is a good technique that deals with such stochastic environments and the multiple objectives can be evaluated by establishing a Pareto dominance relation among themselves.

Thus, Multi-Objective Reinforcement Learning (MORL) method can be thought of as a combination of these two techniques. In reinforcement learning, Q-learning is a technique that is used to find an optimal action-selection policy. In this project, we study the use of voting methods from social choice theory to evaluate the different objectives.

CHAPTER 2

Problem Definition and Motivation

In this section, we will take a look at the problem statement and the scope of this project. We will also describe the motivation behind this work and the contribution to this field of study.

2.1 Problem Definition

The problem of finding an optimal path between two points on a plane can be solved using several algorithms and/or techniques as mentioned earlier in the introduction. However, the study in this project is focused specifically on a reinforcement learning approach to path finding called as Multi-Objective Reinforcement Learning with Voting Q-Learning. This novel approach to many objective path finding was first proposed in [1]. In this project, we evaluate different voting methods such as Copeland voting, Approval voting, etc. and extend the previous work by implementing new voting methods that have not yet been studied in the context of MORL problems to the best of our knowledge. The different voting methods are described in detail in subsequent chapters.

The prior work using VOQL has provided results for 5-Objective deterministic problem and a 5-objective stochastic problem. In this project, we also look at a six-objective deterministic and 6-objective stochastic problem inspired by the benchmark provided in [1]. The project includes the following:

- Study the effectiveness of Voting Q-Learning algorithm for action selection in an environment with many conflicting objectives.

- Implement and analyze results of different voting methods to determine which ones are suitable voting methods.
- Analyze the trend of the required time in seconds per episode and the total reward received per episode.
- Determine a suitable learning rate value by experimentation for the stochastic problem.

2.2 Motivation

Path finding problems can be solved using algorithms like Dijkstra's algorithm, A* search algorithm, Simulated Annealing, etc. But for these algorithms, most of the times it is assumed that there is only one objective to be considered and that the problem environment is fully known in advance and is deterministic. For example, in a simple path finding problem, it is assumed that the agent only wants to reach the goal state in minimum number of steps regardless of any other factors. Also, most of the times the environment is a grid of a fixed size which is known in advance before the algorithm even starts looking for a solution.

However, in a real world scenario, this might not be the case and it is possible that the environment is in fact stochastic and initially unknown. For instance, the agent may not know the grid size of the environment in advance and it is not explicitly instructed about the next action to perform to reach a particular state. In essence, all the agent knows is its start state - S, goal state - G, and the set of other objectives under consideration. And, it is expected to learn the policy π that maximizes the total reward R.

Further, to model real world applications, we may need to consider an environ-

ment with several conflicting objectives. Thus, in part, our problem also becomes a multi-objective optimization problem. The static algorithms like Dijkstra rely on Pareto Dominance to determine the optimal action to perform in a given state. The effectiveness of this Pareto Dominance method decreases as we increase the number of objectives the learning agent should consider. This is because, as we go on increasing the number of objectives, all possible actions become Pareto Optimal and in effect, the actions are selected at random as all the actions are equally dominant or Pareto optimal.

Another challenge in the many objective optimization problems is that, for these problems to be successful, we need to have some form of a priori knowledge about the problem domain. This includes, but is not limited to, the help from domain experts before or during the algorithm execution to guide the learning agent or setting up a predetermined solution preference based on which the agent will model its decisions. In case of help from a domain expert, the process can not be made completely autonomous and thus the usually it is not used on its own but rather used as a supplement to the other methods. Also, this manual intervention by a human entity decreases the scalability of the solution and the system becomes biased to the domain experts preferences.

Also, it is not always feasible to set up a predetermined solution when we do not have a complete idea about the environment. Thus, this method is also ineffective in situations where the end goal or the optimal solution is not known in advance.

All the above points force us to consider an exploratory algorithm such as Reinforcement Learning to find a solution to the path finding problem.All the above points force us to consider an exploratory algorithm such as Reinforcement Learning to find a solution to the path finding problem.

2.3 Contribution

The primary contribution of this project includes evaluation of different voting methods from social choice theory that have not been studied yet in the context of path finding using MORL as an alternative to the Pareto dominance technique. In this project, we also study the existing VOQL results in an attempt to validate the effectiveness of this technique.

This project report is organized in chapters as follows: Chapter 3 defines and elaborates on the terminology used in Reinforcement Learning, Q-Learning, Multi-Objective Reinforcement Learning, Markov Decision Process, and Social Choice Theory. In Chapter 4, we discuss the related work in Multi-Objective Reinforcement learning and Social Choice Theory. Chapter 5 describes the studied problems and the results of VOQL. Finally, in Chapter 6 we conclude with the future work.

CHAPTER 3

Terminology

3.1 Reinforcement Learning

Reinforcement Learning [2] is a machine learning technique in which, given an environment and an artificially intelligent software agent, the goal for the agent is to automatically determine the ideal behavior at each step based on its experience to maximize its performance in the context of that environment.

Unlike supervised learning, in case of Reinforcement Learning, the agent learns from the consequences of its actions. It selects the actions either by exploitation or exploration. After every action in each state, the RL-agent receives a reward. And based on this reward value, the agent tries to learn a policy that maximizes the overall reward.

3.1.1 Definition

Let, the S be the state space, where S = $\{s_1, s_2, ..., s_N\}$, and A be the action space, where A = $\{a_1, a_2, ..., a_r\}$. Then, if the learning agent takes an action a and moves from a state $s \epsilon S$ to another state $s' \epsilon S$ at given time t, then, the the corresponding transition probability is denoted by $T(s'|s, a)$ and the corresponding reward received from the environment is $R(s, a)$ [3].

Policy: The goal of reinforcement learning is to learn a deterministic policy π that maximizes the total reward received from time step t by mapping every state $s \epsilon S$ to an optimal action $a \epsilon A$. This total expected reward from state s by following

the policy π is given by a state-value function

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} | s_t = s\} \tag{1}$$

Where, E_{π}[x] is the expected value of the random variable x when the agent follows the policy π.

By taking an action at each state, the agent moves to a new state and receives some reward from the environment. As the agent explores the environment further more, the policy guides the agent about the action to be taken from each state. In most of the cases this policy can be as simple as a lookup table or it can also be a function that maps a state to an action to be taken or it may be stochastic. The policy is a vital aspect of any reinforcement learning agent. It is sufficient by itself to determine the behavior of the agent.

Reward Function and Reward: As mentioned earlier, when the learning agent moves from one state to the next state in the environment, it receives some reward for taking that action from the environment. This reward is used to update the policy that the agent follows at each state. In any reinforcement learning setting, we need to define a reward function specific to the Reinforcement Learning problem context. Thus the reward function is always contextual because it changes with the change in the problem context. In case of single objective RL problem, the reward function maps each state-action pair to a single numerical value. Thus, the reward is simply a scalar value. The objective of the RL-agent is to maximize the total reward it receives in the long run. This implies that there is a direct correlation between the reward received and the quality of the determined policy.

Value: A reward function determines the immediate next action to be taken in any state. And a value function specifies what is good in the long run in a sense that

it is the total amount of reward the learning agent can expect to receive over the future, starting from the state it is in at that instance of time. Thus, agent receives the reward at each step and uses it along with the value to update the policy.

If the agent follows policy π to move from state s by taking action a, then the value for this action can be represented as $Q^{\pi}(s, a)$. This value is the expected reward for the agent starting in state s, taking action a and following policy π after that. The optimal value $Q^{*}(s, a)$ then

$$Q^{*}(s, a) = R(s, a) + \gamma \sum_{s'} T(s'|s, a) \max_{a'} Q^{*}(s', a') \tag{2}$$

This equation will now lead us into the concept of Q-Learning, a promising, value based Reinforcement learning approach.

3.1.2 Q-Learning

Q-learning algorithm is defined as a model-free reinforcement learning technique. It is called a model-free algorithm, because in order for Q-learning to work it is not necessary to have a fixed model of the environment. The Q-learning algorithm is used to determine a policy for selecting an action in a finite Markov Decision Process [1].

It works by learning an action-value function, often denoted by Q(s,a), which ultimately gives the expected utility of taking a given action a in a given state s, and following an optimal policy thereafter. A policy, often denoted by π, is a rule that the agent follows in selecting actions, given the state it is in.

The Q-learning algorithm was introduced to iteratively approximate the value of Q^{*} given in Equation 2. In this algorithm, a Q-table consisting of values for each state and action pair is stored for lookup. This value stored for each state-action pair is represented as $\hat{\mathrm{Q}}$(s,a). This is the learning agent's estimate of actual Q^{*} value for

a given state action pair. The $\hat{Q}$ value is updated as

$$\hat{Q}(s,a) = (1-\alpha_t)\,.\,\hat{Q}(s,a) \;+\; \alpha_t(r \;+\; \gamma \max_{a'} \hat{Q}(s',a')) \tag{3}$$

Here, r represents the scalar reward value received by the learning agent for taking action a in state s and the value α_t is the learning rate at time t.

The Q-Learning algorithm is given as Algorithm 1:

Algorithm 1: Single-Objective Q-Learning Algorithm

Initialize the environment

begin

Initialize $\hat{Q}$(s,a)

for *each episode e* **do**

Initialize s

while *s is not terminal* **do**

Choose action a to perform in state s using policy derived from the current $\hat{Q}$ values

Perform action a, receive reward r and next state s' from the environment

Use the update rule as follows:

$$\hat{Q}(s,a) = (1-\alpha_t)\hat{Q}(s,a) + \alpha_t(r + \gamma \max_{a'} \hat{Q}(s',a')) \tag{4}$$

Update the current state value to s'

end

end

end

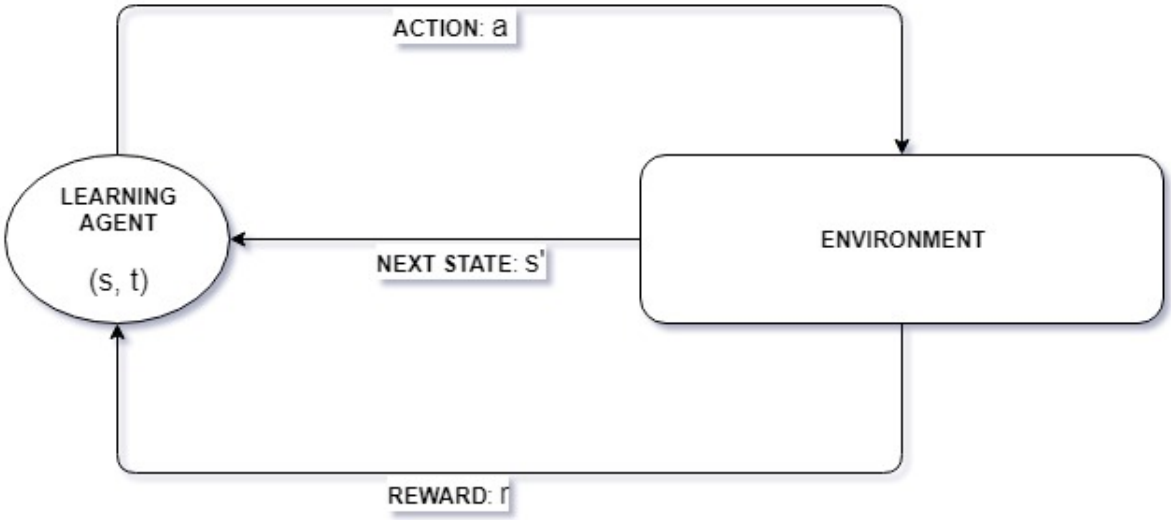

Figure 1: Reinforcement Learning Framework

Here, the Figure 1 describes the single objective reinforcement learning framework. Where the agent, represented by an oval, is in state s at time step t. The agent performs the action a based on the Q-value. The environment receives this action and in response it returns the next state s' to the agent and a corresponding reward r for taking the action a.

3.2 Multi-Objective Reinforcement Learning

Multi-Objective Reinforcement Learning (MORL) is an application of reinforcement learning technique. As the name suggests, MORL is concerned with optimizing the reinforcement learning algorithm for several, possibly conflicting objectives. MORL problems can be modeled as a Multi-Objective Markov Decision Process (MO-MDP) [4]. A MO-MDP can be represented as a tuple T = (S, A, P, R, γ) , where S and A represent the state and action space respectively. This means, the learning agent can be in any state belonging to the set S and can take any action from a set of actions denoted by A. After taking an action a ϵ A at time step t and state s ϵ S, the agent moves in to state s' ϵ S. The probability of this transition from state s to state s' after performing action a, is denoted by $P_a(s, s')$. In case of MO-MDP, the reward R is a little different as compared to the simple Markov Decision Process. The reward is received by the learning agent by transitioning from one state to another by performing some action. This reward R is a vector in which each entry represents the scalar reward value for the corresponding objective [3].

Formally this can be represented as

$$R(s, a) = (R_1(s, a), ..., R_k(s, a)) \tag{5}$$

Here, $R_1, ..., R_k$ are the reward values corresponding to objectives $O_1, ..., O_k$.

And, the value function that depends on the state s at time step t is given as

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \tag{6}$$

Note that, the Equation 6 is a vectorial function as opposed to the scalar function given in Equation 1. This difference is also evident in the RL framework and MORL framework diagrams in Figure 1 and Figure 2 respectively. In the Figure 2, we

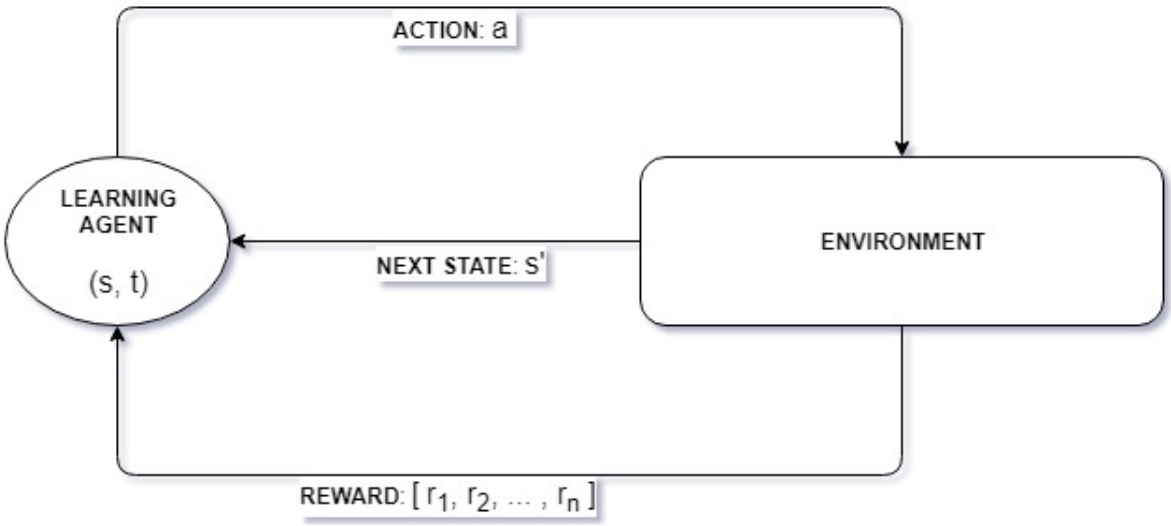

Figure 2: Multi-Objective Reinforcement Learning Framework

describes the multi-objective reinforcement learning framework. Similar to the single-objective RL framework, in MORL framework, the agent is represented by an oval and is in state s at time step t. The agent takes the action a based on the available Q-values. The environment receives this action and in response it returns the next state s' to the agent. But, instead of returning a single scalar value as a reward to the agent for taking the action a, the environment now returns a reward vector R(s, a) as shown in Equation 5.

3.3 Markov Decision Process

Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. Here, we are considering only one state Markov decision process, i.e. the transition to a state s' in the environment only depends on one

previous state s and the action a determined in that state. Thus, the RL problems can be modeled as a one state MDP.

3.4 Social Choice Theory

Social Choice Theory, also known as Voting Theory, is the study of methods for group decision making. Usually, an election is held in which the preferences of each individual voter are aggregated to determine the preference of the entire group [5], [6].

In the election, there are A alternatives available to choose from and there are N voters, each of which will evaluate all the alternatives in set A. After a voter evaluates all the alternatives, a preference ordering R of all the available alternatives is formed for that voter. This is called the ballot of that individual voter and consists of the individual voter's scores for all alternatives in A. Once all the voters complete their voting, all the ballots are collected in a set L. And finally, to complete the election, a social choice function or a social choice correspondence is applied to this set L which produces the outcome of the election. The outcome of the election can be one or more alternatives selected from A based on whether we apply a social choice function or a social choice correspondence.

The outcome of the election is a single alternative in A if we apply a social choice function, or a non-empty subset of A which means we can have more than one alternatives selected as winner candidates.

In this project, we are using social choice function on the set L for action selection. The details of the use of social choice function in MORL are mentioned in subsequent chapters. However, it is worth noting that since we are using social choice function for action selection, in each state, we will strictly have only one action selected as the result of the election between all available actions in that state.

Now, we will define and briefly explain [6] the voting methods that are implemented and evaluated in this project:

3.4.1 Approval Voting

In approval voting, each voter is allowed to vote for one or more alternatives/-candidates and the candidate with the highest number of votes wins. This essentially means that the voters can endorse all the candidates they approve of and thus it is different from the traditional plurality method. One thing to note in Approval method is that voters can also abstain from voting, meaning a voter can choose not to vote for any candidate.

The approval voting algorithm is as shown below in Algorithm 2

Algorithm 2: Approval Voting Algorithm

Data: List of all available candidates

Result: List of one or more candidate(s)

Initialize votes(k) = 0 for all candidates k

begin

  for *each voter v* **do**

    for *each candidate k* **do**

      if *v approves of k* **then**

        votes(k) = votes(k) + 1

      end

    end

  end

  Return candidate k as winner, where votes(k) = max(votes)

end

3.4.2 Borda Count

In the Borda count method, each voter needs to provide an ordering of the available candidates. Each voter assigns the Borda score to all the candidates. The Borda score is assigned as follows:

Assuming that there are n available candidates, then the candidate ranked first is assigned n-1 points, candidate ranked second is assigned n-2 points, ... , the second to last candidate is assigned 1 point, and finally, the least preferred candidate is assigned 0 points. Once all the voters are done voting, a cumulative Borda score,

B(candidate), for each candidate is calculated using the following formula

B(candidate) = (n-1) * {i | i selected candidate as first} +
(n-2) * {i | i selected candidate as second} + ... +
(1) * {i | i selected candidate as 2nd to last} +
(0) * {i | i selected candidate as last}

The candidate with the highest Borda count at the end of the election is declared as the winner.

The Borda count algorithm is as shown below in Algorithm 3

Algorithm 3: Borda Count Method

Data: List of all available candidates

Result: One optimal candidate

Assume total number of candidates to be n.

begin

for *each voter v* **do**
Arrange all n candidates according to the preference ordering

end

BS(k) = (n-1) * {i | i selected k as first preference} +
(n-2) * {i | i selected k as second preference}+ ... +
(1) * {i | i selected candidate as 2nd to last preference} +
(0) * {i | i selected candidate as last preference}

Return candidate k as winner, where BS(k) = max(BS $\forall\, candidates\, k$)

end

3.4.3 Copeland Voting

A pairwise election is conducted between all the available candidates by each of the voters. The winner is selected by the majority vote method at the end of the election. In order to conduct the election, the voters rank the candidates in an arbitrary order. And then compare each candidate with the others. A candidate is rewarded one point after winning a pairwise election. Both candidates are given 1/2 points for a tied election. And the candidate receives 0 points in case it is defeated in an election. Once all the voters have completed a pairwise election process explained above, a cumulative score is calculated by summing all the individual scores received from the pairwise elections. A candidate with highest cumulative score is declared the winner at the end of election.

The Copeland voting algorithm is as shown below in Algorithm 4

Algorithm 4: Copeland Voting Method

Data: List of all available candidates

Result: One optimal candidate

Assume total number of candidates to be n. Initialize score(k) = 0 for all candidates k

begin

  for *each voter v* **do**

    for *each candidate k* **do**

      Conduct pairwise election with all other n-1 candidates

      if *k is winner of pairwise election* **then**

        score(k) = score(k) + 1

      end

      else if *The election is a draw* **then**

        score(k) = score(k) + 0.5

      end

    end

  end

  Return candidate k as winner, where score(k) = max(score $\forall$ *candidates k*)

end

3.4.4 Negative Voting

Negative voting is a stricter form of the earlier Approval method in a sense that a voter can either approve of a candidate or can disapprove while voting. In case of

negative voting, a voter can choose to vote for a candidate and reward 1 point to the candidate or the voter can choose to vote against the candidate and give the candidate -1 points. In case of negative voting there could be more than one winners of the election. The only difference in Approval and Negative voting is that in Negative voting, the voters can select only one candidate to vote for or vote against. On the other hand, in case of Approval voting, voters can select any subset of candidates to approve or disapprove.

The Negative voting algorithm is as shown below in Algorithm 5

Algorithm 5: Negative Voting Method

Data: List of all available candidates

Result: List of one or more candidate(s)

Assume total number of candidates to be n. Initialize score(k) = 0 for all candidates k

begin

  for *each voter v* **do**

    Select one candidate k to vote for or vote against.

    if *v choose to vote for k* **then**

      score(k) = score(k) + 1

    end

    else if *v choose to vote against k* **then**

      score(k) = score(k) - 1

    end

  end

  Return candidate k as winner, where score(k) = max(score $\forall$ *candidates k*)

end

3.4.5 Plurality Rule

Finally, we will define one of the easiest voting methods which is still widely used because of its simplicity. In Plurality rule, each voter can select at max one candidate. For N candidates, the ballot of an individual voter would include a preference ordering of all the candidates out of which only the topmost candidate is of interest for that

voter. Ballots of all the voters are collected and the candidate with the highest number of top votes is elected as the winner.

The Plurality voting algorithm is as shown below in Algorithm 6

Algorithm 6: Plurality Voting Method

Data: List of all available candidates

Result: One optimal candidate

Assume total number of candidates to be n. Initialize votes(k) = 0 for all candidates k

```
begin
    for each voter v do
        for each candidate k do
            if k is preferred then
                votes(k) = votes(k) + 1
                break
            end
        end
    end
    Return candidate k as winner, where votes(k) = max(votes ∀ candidates k)
end
```

Now, we can take a look at the VoQL algorithm proposed in [1], in which we can

use the voting methods mentioned above.

Algorithm 7: VoQL Algorithm

Data: Voting Method, Learning Rate α, Discount Factor γ

Result:

Select a voting method for action selection;

Initialize $Q_{set}(s, a)$ *as empty sets* $\forall\, s \in S$, *and* $a \in A$) *Initialize the learning rate parameter* α *and discount factor* γ

begin

for *each episode e* **do**

Set s as the initial state

while *s is not terminal state* **do**

Transform set of all optimal Q(s,a) values for each potential action into ballots for each objective

Choose action a by holding election using ballots for each objective with the selected voting method

Store the reward vector r^T and next state s' for taking the action a

Calculate optimal action a'_{max} from s'

Calculate the set of non-dominated vectors ND($Q_{set}(s', a'_{max})$) using the selected voting method

Use the following update rule to update the Q-Table

$Q_{set}(s, a) = Q_{set}(s, a) + \alpha * [r^T + \gamma * Q_{set}(s', a'_{max}) + Q_{set}(s, a)]$

Set s = s'

end

end

end

CHAPTER 4

Related Work

4.1 Previous Work in MORL

In Multi-Objective Reinforcement Learning problems, the learning agent receives a vector of reward values consisting of reward for each of the objectives under consideration. The initial research on solving the MORL problems focused on converting the vector reward to a single scalar value through a process called as scalarization of the vectors. The scalarization function is a linear function that maps a given vector to a scalar value as shown by Natarajan and Tadepalli in [7]. However, it can also be non-linear as demonstrated in [8] by Gábor et al. The scalarization function will eventually output a scalar reward value. Once the reward vectors are converted to a single scalar value, traditional RL approach can be applied to solve the problem. Though very simple and effective in cases where the multiple objectives are correlated, the scalarization process can be ineffective in a lot of real world cases where the objectives are not always correlated [9].

Action-selection is an important part of solving any RL problem. In case of a single-objective Reinforcement Learning problem, it is very straightforward to select an action based on the reward received in each state. An action can be selected for every state by comparing all the values for that state and selecting the action with the maximum value. However, in case of multi-objective Reinforcement Learning problem, modeled as a MO-MDP, selecting an action is not as straightforward as single-objective MDP. Here, the values for state-action pair are represented as a vector of values corresponding to all the objectives. So in order to select a dominant action, a dominant vector needs to be selected from all available alternatives.

A lot of the previous work on MORL has shown the effectiveness of using Pareto dominance for action selection. In [10], the authors of the paper described an algorithm for linear value function approximation with an example of 3-reward function case. This approach was based on Q-learning as compared to some of the other work in [11], which is based on value iteration and in [12], which is based on policy iteration.

A comprehensive survey of multi-objective sequential decision making algorithms was presented by Vamplew et al. [13]. This study identified two classes of algorithms for multi-objective Reinforcement Learning. The two classes identified were Single Policy Algorithms and Multiple-Policy Algorithms. In single policy algorithms, only one best fit policy is selected based on the users input or domain knowledge of the problem environment. On the other hand, the algorithms that try to find a set of policies in order to approximate the Pareto front were categorized as multiple-policy algorithms.

The previous research is not only limited to Reinforcement learning, rather one of the successful approach to MORL problem is based on multi-objective Monte Carlo Tree Search algorithm using hypervolume quality indicator parameter [14]. The effectiveness of hypervolume-based multi-objective reinforcement learning (HB-MORL) over linear scalarization was again confirmed in the same year [15]. The authors of the paper concluded that HB-MORL outperforms linear scalarization method by a margin and is especially effective in situations where the algorithm can not be tuned in advance for a particular problem instance.

Other approaches to solving MORL problems involved the use of simulation based genetic algorithm for a stochastic road network [16], application of dimensionality reduction technique to exact label-setting multi objective search algorithm in order to reduce the number of dominance checks and get faster results [17], use of Artificial

Immune System (AIS), chaos operator, and Particle Swarm Optimization (PSO) [18], etc. It was shown that the last approach outperforms GA and PSO while considering the optimality of the route and convergence time as the metrics.

4.2 Previous Work in Social Choice Theory

Social Choice theory is the study of different voting methods that are used for group decision making. One of the simplest method for selecting an alternative as a group from all available choices is that each individual in the group is allowed to select only one alternative and then the winner is the alternative that was selected by most number of voters. However, there are several disadvantages to the plurality voting method. The credit for the earliest work in Social Choice Theory for providing mathematical definitions of new voting methods to improve upon the drawbacks of plurality methods goes to Borda [19] and Condorcet [20]. Following these studies, Arrow formalized the study of social choice theory. He introduced four conditions that an ideal voting method in Social Choice theory would satisfy, which are Pareto efficiency, independence of irrelevant alternatives, unrestricted domain, and non-dictatorship [21].

An interesting approach intersection MORL and Social Choice Theory was proposed and demonstrated in [1]. The authors of the paper proposed an innovative algorithm called as Voting Q Learning that uses the voting methods from Social Choice Theory for action-selection in multi-objective Reinforcement Learning problem. They also introduced a benchmark problems for MORL which we evaluate in the further sections in this paper.

CHAPTER 5

Methodology and Experimental Results

5.1 Environment Setup

To start the experiments we set up a 5 * 5 grid environment with following entities as part of the environment.

1. Learning Agent:

   The learning agent is represented with a white square box located at the top left corner of the grid. This is the initial state of the agent, which is the only moving entity in the environment. Once the agent is activated and is ready to move, it will change to a pink square box.

2. Goal State:

   Goal state is represented with a green circle at the center of the grid. Once the learning agent reaches this location the current episode terminates and the environment is reset to its original state.

3. Hell State:

   The hell state is shown in the gird with a red square box. There are 2 hell states in the environment and the learning agent should avoid reaching the hell state.

4. Adversary:

   The adversary is represented with the orange square at the top right corner of the environment. The adversary is stationary and the agent can also be in the same position as the adversary.

5. Monitoring Tower:

   A virtual and invisible monitoring tower is located at the origin position of the agent i.e. at the top left corner of the grid. This tower is to monitor the movements of the agent and to maintain connectivity with the agent.

5.2 Environment States and Agent Movements

As shown in the Figure 3, there are two main states in the environment. The initial state and the goal state. Each episode starts with the agent in the initial state and ends when the agent reaches the goal.

There are four possible actions allowed for the agent: move left, move right, move up, and move down. If the agent tries to move to an unavailable state, the environment returns a zero reward for every objective and the agent does not move i.e. there is no change in the Q-table for that iteration. An example of this scenario is shown in Figure 4 (B), where the agent is only allowed to move up, down, or right. If the agent tries to move left, the environment will not allow this move and agent will remain in the same state.

Other intermediate states are also shown in Figure 4. Note that, the agent continues to move towards the goal state even when it reaches the hell state first i.e. the episode does not terminate unless the agent reaches the goal state.

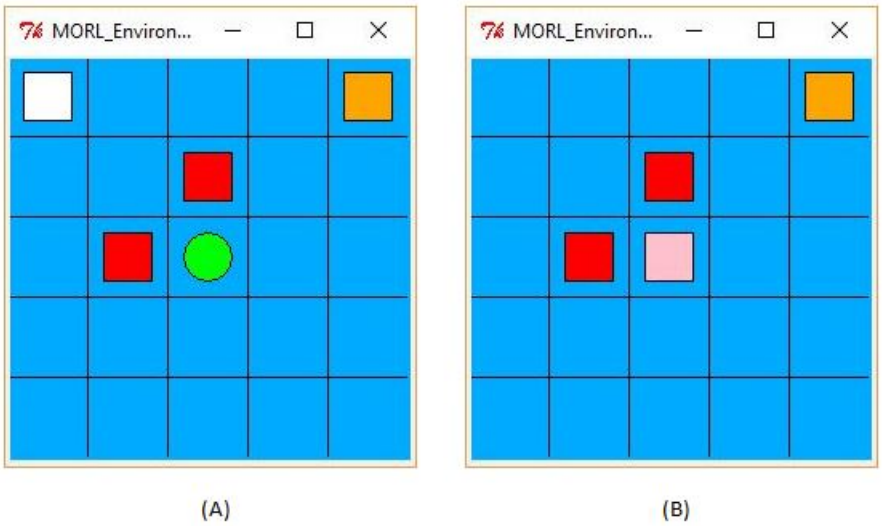

Figure 3: Environment Main States
(a) Initial State, (b) Goal State

5.3 VoQL Evaluation

We implemented the VoQL algorithm given in Algorithm 7 using Approval voting, Borda Count method, Copeland voting, Negative voting, and Plurality method to evaluate the performance of these different voting methods. The problem stated above is a benchmark problem for many-objective problems and was first presented in [1]. However, to extend the previous work we added the evaluation of Negative voting and Plurality method as shown in Algorithms 5, and 6 respectively that were not studied previously. Also, a new objective to avoid hell-state while aiming to reach the goal was also introduced on top of the five other objectives. The computer simulations were run for a 6 objective deterministic problem on a 5 x 5 grid and 6 objective stochastic problem on a 5 x 5 grid, the details of which are mentioned in the subsequent sections in this chapter.

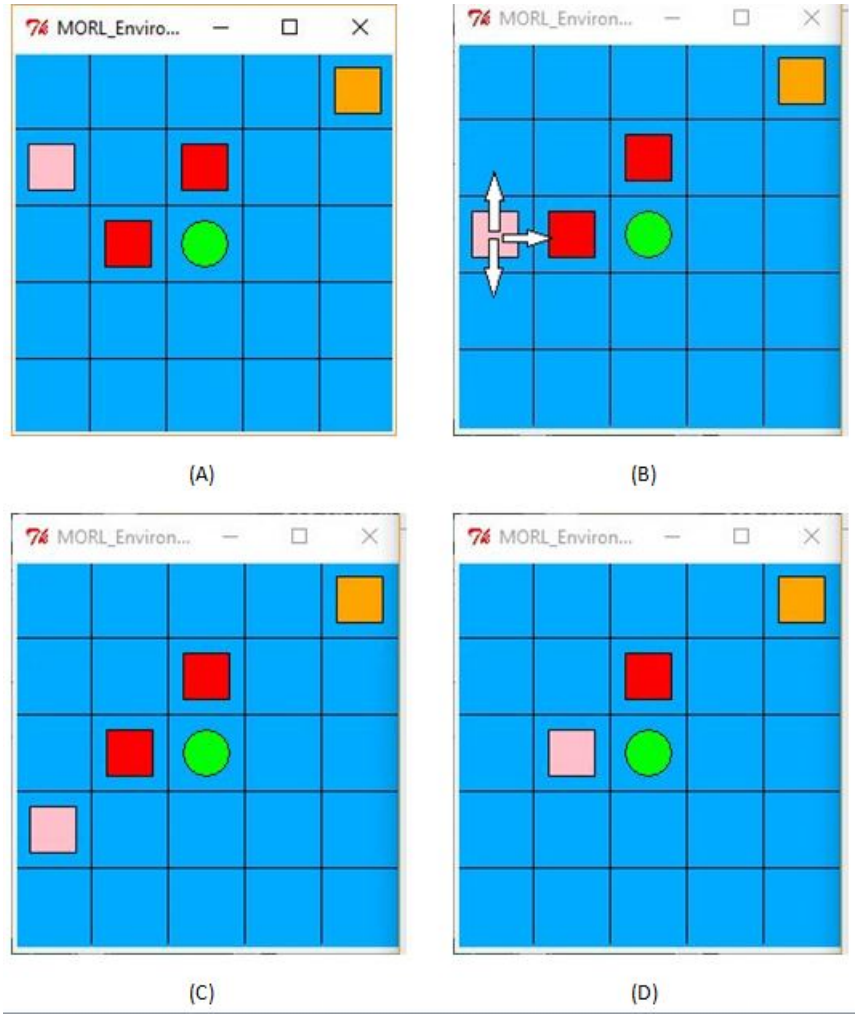

Figure 4: Environment Intermediate States

5.4 6-Objective Deterministic Problem

Building on top of the 5-objective deterministic problem in [1], we have added an extra objective. The 6 objectives that are considered for this problem are:

1. Reach the goal, Avoid Obstacles
2. Minimize the number of steps required to reach the goal
3. Avoid the adversary at the top right corner of the grid

4. Minimize the time required to reach the goal state

5. Minimize the amount of fuel required to reach the goal

6. Minimize the signal loss to monitoring tower located at the origin

We present below the reward for each of these 6 objectives. The value in each cell of the grid represent the numerical reward the learning agent receives in each state of the grid for that reward.

0	0	0	0	0
0	0	-5	0	0
0	-5	6	0	0
0	0	0	0	0
0	0	0	0	0

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

0	-3	-4	-5	-6
0	-2	-3	-4	-5
0	-1	-2	-3	-4
0	0	-1	-2	-3
0	0	0	-1	-2

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-2	-2	-2	-2	-2
-3	-3	-3	-3	-3

-1	-1	-1	-1	-5
-1	-1	-1	-1	-5
-1	-1	-1	-1	-5
-2	-2	-2	-2	-5
-3	-3	-3	-3	-8

0	-1	-2	-3	-4
-1	-2	-3	-4	-5
-2	-3	-4	-5	-6
-3	-4	-5	-6	-7
-4	-5	-6	-7	-8

We are taking the reward values as negative because all our objectives under consideration are for minimization of the total reward, whereas, the reinforcement learning problems involve maximization of the total expected reward for the learning agent. Thus, we want the agent to earn the least negative reward while exploring the environment to reach to the goal state.

We ran 30 iterations of 1000 episodes for each method and then averaged the reward received for each episode and time taken to complete each episode to have statistically significant result values. The 30,000 runs of the same action selection generated sufficient data to compare the different voting methods.

Experiment 1 - different voting methods : episode vs time (sec.)
We used time required to complete each episode and the total reward per episode as the performance evaluation metrics. The averaged time and reward values are good evaluation metrics to compare the performance of individual voting method over time as the agent learns. Along with that, the time and reward values per episode give a good way to compare the performance of these different methods with one another.

The scatter plot in Figure 5 is the time required in sec. per episode. To obtain a smooth trend of the time required to reach the goal state over the episodes, we

averaged the time for every 20 episodes. As we can see in the scatter plot, the time required to complete an episode goes on decreasing significantly and then at around 400 episodes the time delta becomes smaller and smaller. This is when the agent converges on an optimal path to the goal state. We can also look at the performance of the five voting methods in comparison to one another. We can see that the performance of Copeland, Borda, and Negative voting methods is almost the same on an average. Following these methods, Approval voting registered higher average time. Finally, as expected Plurality method took the longest to converge and registered the highest time required per episode on an average.

This similar trend can be seen in Figure 6, which is a bar graph comparing the average time required for every 100 episodes. It is evident that plurality method has the highest average time required, followed by Approval voting and the other voting methods.

Experiment 2 - different voting methods : episode vs reward

Finally, for the 6-objective deterministic problem we compared the average values of the total reward received by each voting method. The reward vectors were converted to a scalar value by calculating the magnitude value of the vectors and then summing them together. The expected result is to get reward with minimum magnitude as the actual reward values are negative values.This represents the minimization goal of the objectives under consideration.

The trend of decreasing reward magnitude can be seen in Figure 7, where we can see that the Plurality voting and Approval voting methods have the highest reward values throughout the 1000 episodes. On the other hand, the other three voting methods, show a comparable reward value trend.

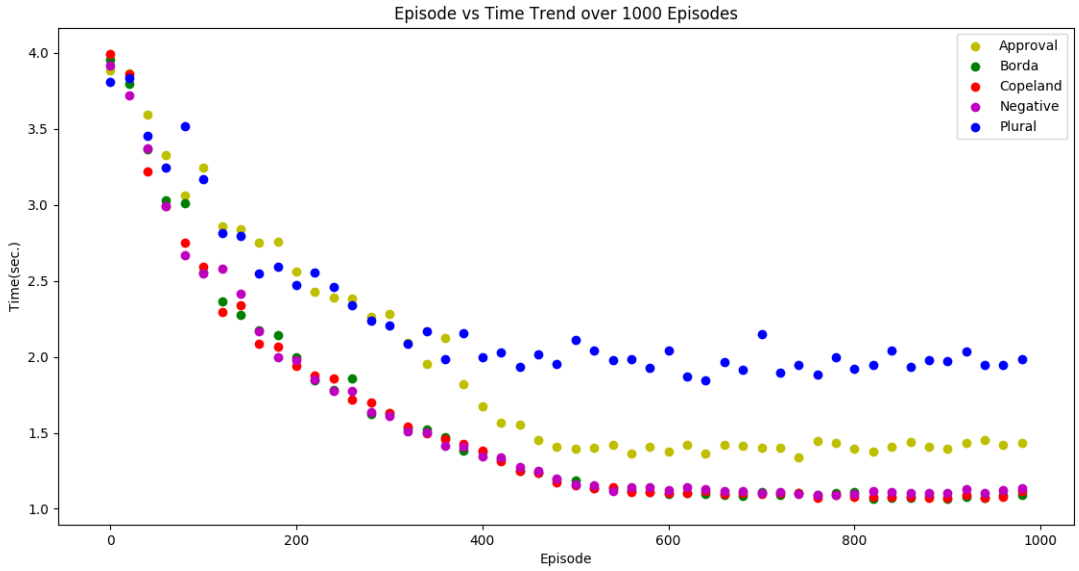

Figure 5: Deterministic 6 Objective Problem - Episode vs Time Per Episode (Sec.)

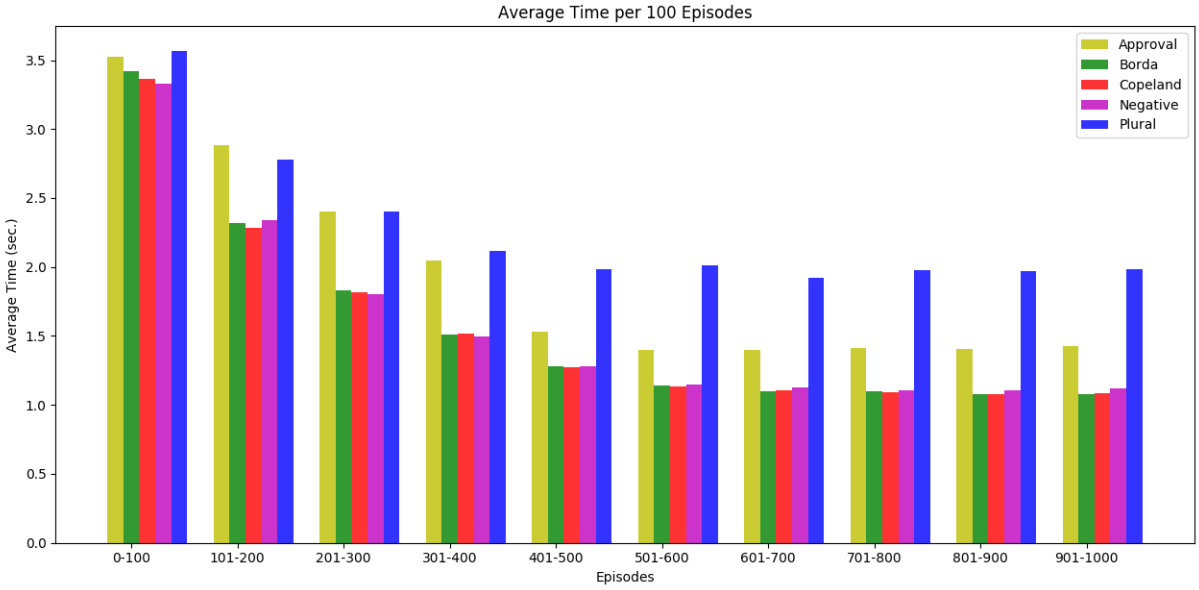

Figure 6: Deterministic 6 Objective Problem - Episode vs Avg. Time Per 100 Episodes

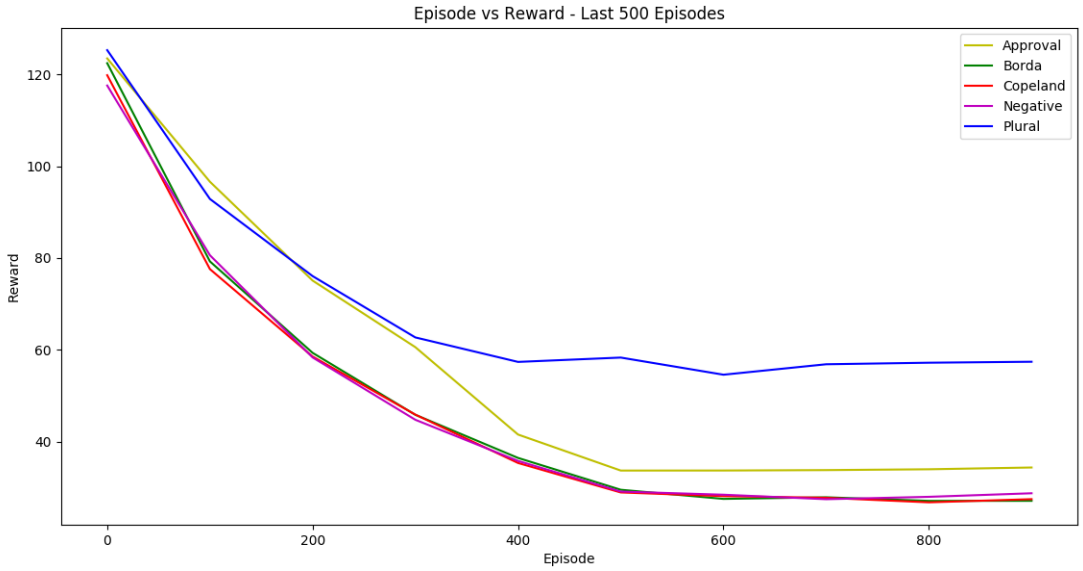

Figure 7: Deterministic 6 Objective Problem - Episode vs Reward Per 100 Episodes

5.5 6-Objective Stochastic Problem

To convert the above problem to stochastic MORL problem, we introduced a probability matrix for the objective of minimizing the signal loss to the monitoring station located at the origin. The value specifies the probability of receiving the reward for that grid cell. The probability matrix is as shown below:

0%	10%	20%	30%	40%
10%	20%	30%	40%	50%
20%	30%	40%	50%	60%
30%	40%	50%	60%	70%
40%	50%	60%	70%	80%

As opposed to the 6-objective deterministic problem, in case of this 6-objective

stochastic problem, we determined the best suitable learning rate value through experimentation. The experiment results are shown here in Figure 8. In the above 3-D plot, we can see that the learning rate is plotted on x-axis, time required per episode on y-axis, and the episode number on z-axis. We expect the time required to reach the goal state to be minimum, thus we are looking for a learning rate value that shows lowest values on the y-axis. As it can be seen that the for the range of learning rate value between 0.7 - 0.9, the average time required per episode is minimum. Thus, we selected the value of learning rate $\alpha = 0.8$.

The results shown below are displaying a trend that with the increase in the number of episodes the reward decreases and so does the time required to reach the goal state also decreases. So, we can conclude that the deterministic and stochastic problems have the similar trend for reward vs episode and average time vs episode plots. However, it is worth noting that both the reward value and the time required are less on average as compared to the deterministic problem.

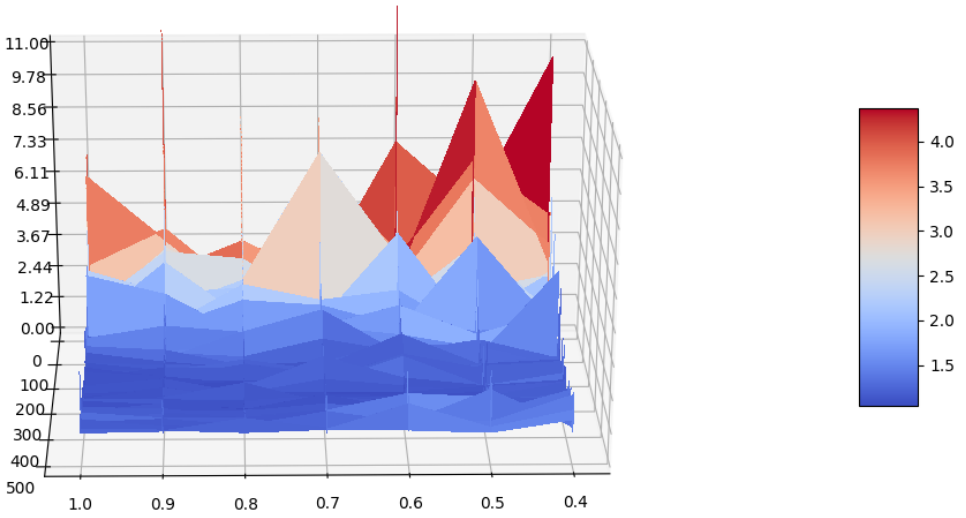

Figure 8: Stochastic 6 Objective Problem - Learning Rate Selection

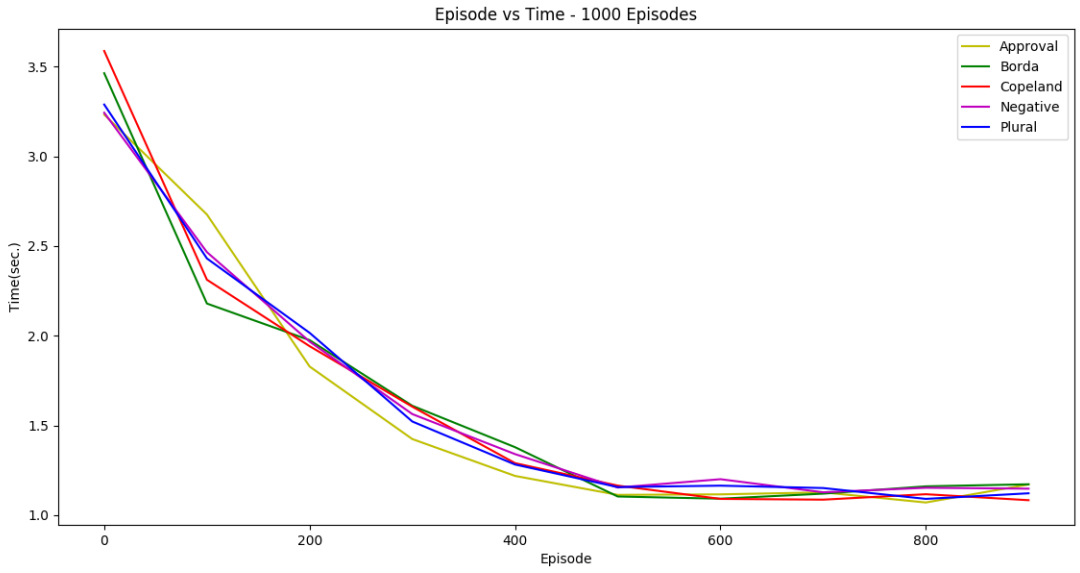

Figure 9: Stochastic 6 Objective Problem - Episode vs Time Per Episode (Sec.)

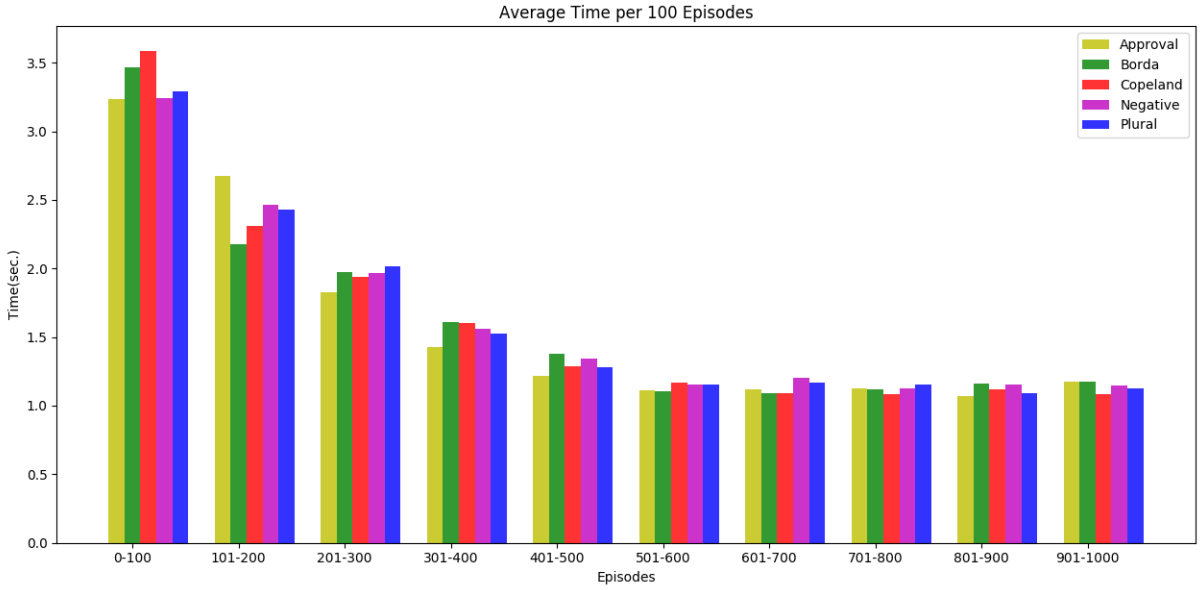

Figure 10: Stochastic 6 Objective Problem - Episode vs Avg. Time Per 100 Episodes

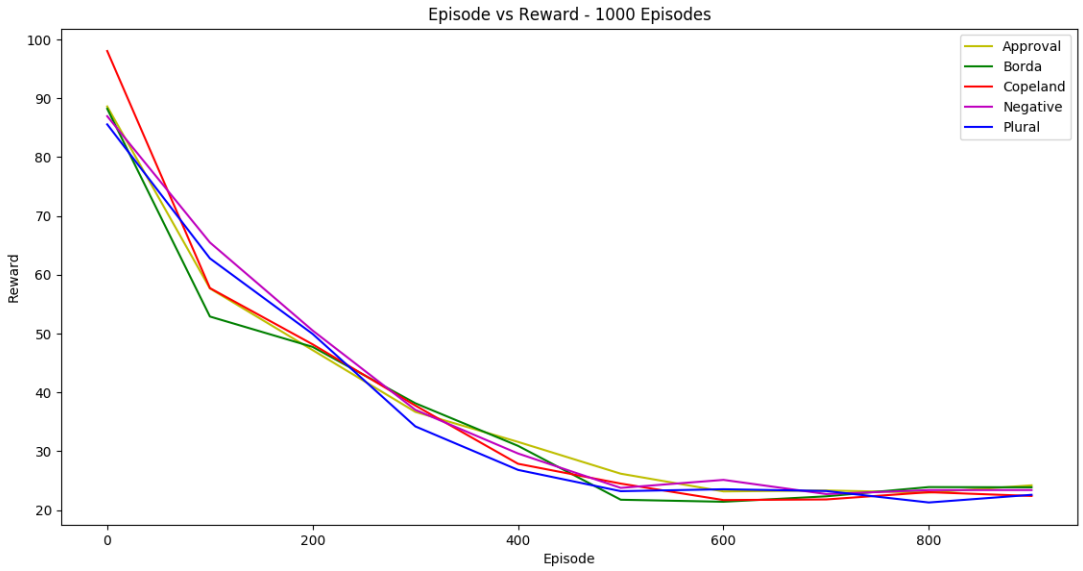

Figure 11: Stochastic 6 Objective Problem - Episode vs Reward Per Episode (Sec.)

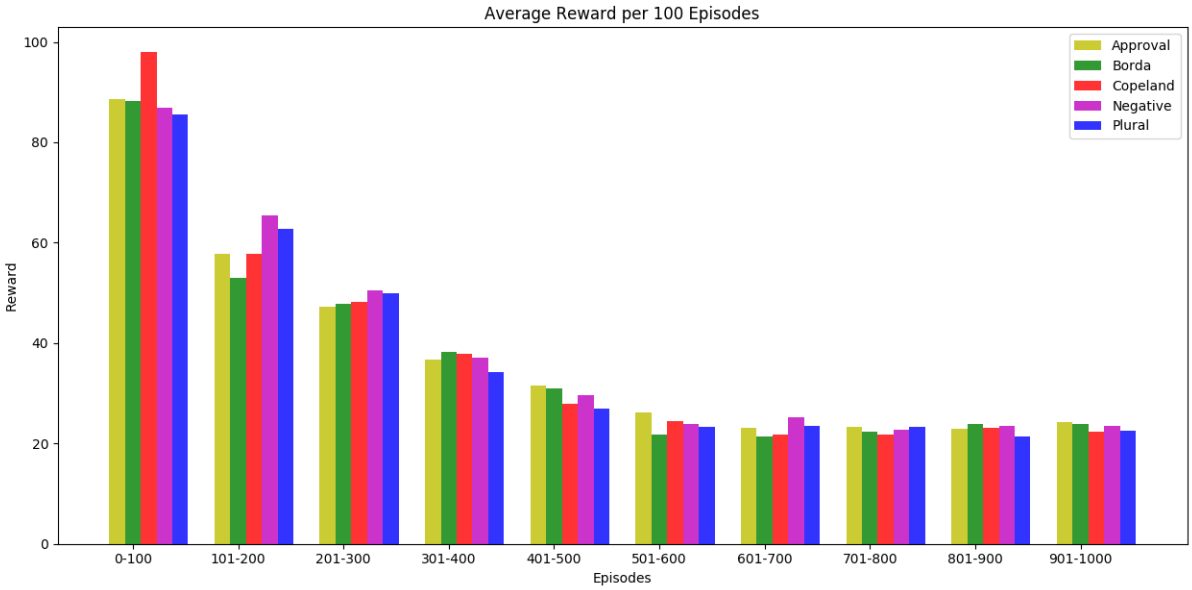

Figure 12: Stochastic 6 Objective Problem - Episode vs Avg. Reward Per 100 Episodes

CHAPTER 6

Conclusion and Future Work

In this project, we evaluated the effectiveness of VoQL algorithm for multi-objective path finding problems where the agent is allowed to move in the environment, receiving reward for each action in order to determine an optimal policy in an environment with many conflicting objectives. We also explored different voting methods to be used with the VoQL algorithm such as Approval voting, Borda Count method, Copeland voting, Negative voting, and Plurality voting. The later two voting methods, Negative and Plurality voting were evaluated for the first time in the context of VoQL algorithm for action-selection in a reinforcement learning problem.

The performance was evaluated using a 6-objective deterministic and stochastic problem. The quality of the solution was determined by using total reward for all the objectives and average time required per episode as the performance evaluation metrics. The results showed that Copeland voting, Negative voting, and the Borda count method performed similar. Following them, Approval voting method recorded second highest average time per episode. Finally, Plurality voting method showed the highest average time required to complete an episode. The total reward value comparison showed the similar trend in performance for these voting methods, with Copeland voting method performing the best and Plurality method showing the least effective results.

In this project, we considered the action space to be limited to only four allowed actions. However, in the future, it would be interesting to look at the problems with larger state and action space. It would be interesting to see how the agent behaves

using VoQL when there is a higher density of number of obstacles in the path towards the goal state. Applications of this algorithm on a complex real-world problem can be more meaningful as opposed to the computer simulated grid-based problems evaluated in this project.

LIST OF REFERENCES

[1] B. Tozer, T. Mazzuchi, and S. Sarkani, "Many-objective stochastic path finding using reinforcement learning," *Expert Systems with Applications*, vol. 72, no. Supplement C, pp. 371 – 382, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417416305863

[2] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[3] K. Van Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of pareto dominating policies," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3483–3512, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2750356

[4] K. Chatterjee, R. Majumdar, and T. A. Henzinger, "Markov decision processes with multiple objectives," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 2006, pp. 325–336.

[5] A. Sen, "Chapter 22 social choice theory," ser. Handbook of Mathematical Economics. Elsevier, 1986, vol. 3, pp. 1073 – 1181. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1573438286030047

[6] S. J. Brams and P. C. Fishburn, "Chapter 4 voting procedures," in *Handbook of Social Choice and Welfare*, ser. Handbook of Social Choice and Welfare. Elsevier, 2002, vol. 1, pp. 173 – 236. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157401100280008X

[7] S. Natarajan and P. Tadepalli, "Dynamic preferences in multi-criteria reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 601–608.

[8] Z. Gábor, Z. Kalmár, and C. Szepesvári, "Multi-criteria reinforcement learning." in *ICML*, vol. 98, 1998, pp. 197–205.

[9] P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry, "On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts," pp. 372–378, 12 2008.

[10] D. J. Lizotte, M. Bowling, and S. A. Murphy, "Linear fitted-q iteration with multiple reward functions," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 3253–3295, Nov. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2503346

[11] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 41–47. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390162

[12] S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli, "Policy gradient approaches for multi-objective sequential decision making: A comparison," 01 2014.

[13] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *Journal of Artificial Intelligence Research*, vol. 48, pp. 67–113, 2013.

[14] W. Wang and M. Sebag, "Hypervolume indicator and dominance reward based multi-objective monte-carlo tree search," *Machine learning*, vol. 92, no. 2-3, pp. 403–429, 2013.

[15] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Hypervolume-based multi-objective reinforcement learning," in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2013, pp. 352–366.

[16] Z. Ji, A. Chen, and K. Subprasom, "Finding multi-objective paths in stochastic networks: a simulation-based genetic algorithm approach," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 1, June 2004, pp. 174–180 Vol.1.

[17] F.-J. Pulido, L. Mandow, and J.-L. P. de-la Cruz, "Dimensionality reduction in multiobjective shortest path search," *Computers & Operations Research*, vol. 64, pp. 60 – 70, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054815001240

[18] Y. Zhang, Y. Jun, G. Wei, and L. Wu, "Find multi-objective paths in stochastic networks via chaotic immune pso," *Expert Systems with Applications*, vol. 37, no. 3, pp. 1911–1919, 2010.

[19] H. Young, "An axiomatization of borda's rule," *Journal of Economic Theory*, vol. 9, no. 1, pp. 43 – 52, 1974. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022053174900738

[20] [Online]. Available: http://www.jstor.org/stable/1961757

[21] K. J. Arrow, *Social Choice and Individual Values*. Yale University Press, 1963, no. 12.