

**SWARM AND PHEROMONE BASED REINFORCEMENT  
LEARNING METHODS FOR THE ROBOT(S) PATH  
SEARCH PROBLEM**

**DISSERTATION**

**SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
AWARD OF THE DEGREE**

**OF**

**MASTER OF TECHNOLOGY  
IN  
CONTROL & INSTRUMENTATION**

**SUBMITTED BY:**

**NUPUR JHA**

**(2K13/ C&I/ 08)**

**UNDER THE SUPERVISION OF**

**DR. BHARAT BHUSHAN**



**DEPARTMENT OF ELECTRICAL ENGINEERING**

**DELHI TECHNOLOGICAL UNIVERSITY**

**(Formerly Delhi College of Engineering)**

**Bawana Road, Delhi-110042**

**INDIA**

**2015**

# DEPARTMENT OF ELECTRICAL ENGINEERING

## DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

### CERTIFICATE

I, **Nupur Jha**, Roll No. 2K13/C&I/08, a student of M. Tech. (Control & Instrumentation), hereby declare that the dissertation titled “Swarm and Pheromone based Reinforcement Learning methods for the Robot(s) Path Search Problem” is a bonafide record of the work carried out by me under the supervision of **Dr. Bharat Bhushan** of Electrical Engineering Department, Delhi Technological University in partial fulfilment of the requirement for the award of the degree of Master of Technology and has not been submitted elsewhere for the award of any other Degree.

Place: Delhi

( **Nupur Jha** )

Date:

DR. BHARAT BHUSHAN

**SUPERVISOR**

Associate Professor

Electrical Engineering Department

Delhi Technological University

Delhi - 110042

## ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Bharat Bhushan for his guidance and assistance in the thesis. Without his consistent support, encouragement and valuable inputs, this project would not have been possible.

I would like to express my deep gratitude to Prof. Madhusudan Singh, the HoD of Electrical Engineering Department without his moral support my project would not have reached to this level.

I would also like to thank my batch-mates and friends who have encouraged and helped me in completing the thesis work. Finally, I express my deep sincere thanks to my Parents who were always there in times of need.

NUPUR JHA

( 2K13/C&I/08 )

M. Tech. ( C&I )

Delhi Technological University, Delhi

## ABSTRACT

With the world moving to an automated platform, robots are finding application in almost all domains to reduce the human effort. One such domain is to path a find in an unknown and hostile environment to reach the goal. The complexity of many tasks arising in this domain makes it difficult for the robots (agents) to solve this with pre-programmed agent behaviours. The agents must, instead, discover a solution on their own, using learning.

In ordinary reinforcement learning algorithms, a single agent learns to achieve a goal through many episodes. If a learning problem is complicated or the number of agents is more, it may take more computation time to obtain the optimal policy and sometimes may not be able to reach the goal. Meanwhile, for optimization problems, multi-agent search methods such as particle swarm optimization, ant colony optimization have been recognized to find rapidly a global optimal solution for multi-modal functions with wide solution space.

This thesis work proposes a SARSA based reinforcement learning algorithm using multiple agents where the agents are guided by the pheromone levels also called the Phe-SARSA. In this algorithm, the multiple agents learn through not only their respective experiences but also with the help of pheromone trail left by other agents to search for the shortest path. The algorithms have been simulated in the MATLAB 2013a and the results have been compared with the Q-learning, SARSA, Q-Swarm, SARSA-Swarm and Phe-Q algorithms.

## TABLE OF CONTENTS

<b>CERTIFICATE.....</b>	<b>i</b>
<b>ACKNOWLEDGEMENT.....</b>	<b>ii</b>
<b>ABSTRACT.....</b>	<b>iii</b>
<b>CONTENTS.....</b>	<b>iv</b>
<b>LIST OF FIGURES.....</b>	<b>vii</b>
<b>LIST OF TABLES.....</b>	<b>xiii</b>
<b>CHAPTER 1. INTRODUCTION.....</b>	<b>1</b>
1.1 Overview.....	1
1.2 Reinforcement Learning.....	2
1.3 Path Search and RL.....	2
1.4 Ant Colony Optimization (ACO).....	3
1.5 Particle Swarm Optimization (PSO) & Group Effort.....	4
1.6 Objective of Work.....	4
1.7 Organization of Thesis.....	5
<b>CHAPTER 2. LITERATURE REVIEW.....</b>	<b>6</b>
2.1 Introduction.....	6
2.2 Brief Review of Papers.....	6
<b>CHAPTER 3. REINFORCEMENT LEARNING.....</b>	<b>16</b>
3.1 Biological Inspiration.....	16
3.2 Introduction.....	16
3.3 Elements & Basic RL framework.....	17
3.3.1 Policy.....	17
3.3.2 Reward Function.....	17
3.3.3 Value Function.....	18
3.3.4 Basic Framework of RL.....	18

3.4	Markov Decision Process.....	19
3.5	Temporal Difference (TD) Learning.....	21
3.5.1	TD Prediction.....	21
3.5.2	Optimality of TD(0).....	23
3.5.3	Value Functions & the Bellman Equations.....	24
3.5.4	Q – Learning: Off- Policy TD Control.....	26
3.5.4.1	Model-based Q-iteration algorithm.....	26
3.5.4.2	Model-free value iteration and the need for exploration...	28
3.5.5	SARSA: On – Policy Control.....	30
3.6	Limitations of RL.....	32
	<b>CHAPTER 4. MULTI-AGENT GRIDWORLD PROBLEM.....</b>	<b>34</b>
4.1	Grid World Problem.....	34
4.1.1	Single Agent Problem.....	35
4.1.2	Multi Agent Problem.....	36
4.2	Multi-Agent Reinforcement Learning (MARL).....	36
4.2.1	Friend Or Foe Algorithm.....	37
	<b>CHAPTER 5. HYBRID RL.....</b>	<b>40</b>
5.1	Introduction.....	40
5.2	RL and Particle Swarm Optimization (PSO).....	40
5.2.1	Q – Swarm.....	40
5.2.2	SARSA – Swarm.....	42
5.3	RL and Ant Colony Optimization (ACO).....	43
5.3.1	Pheromone – Q.....	44
5.3.1.1	Belief Factor.....	45
5.3.2	Pheromone – SARSA.....	47
	<b>CHAPTER 6. SIMULATION RESULT &amp; DISCUSSION.....</b>	<b>50</b>

6.1	Single Agent Problem.....	51
6.1.1	Case I: Obstacles: Fixed; Goal: Fixed.....	51
6.1.2	Case II: Obstacles: Fixed & Moving (Both); Goal: Fixed.....	55
6.1.3	Case III: Obstacles: Fixed; Goal: Moving.....	58
6.1.4	Case IV: Obstacles: Fixed & Moving (Both); Goal: Moving...	62
6.2	Two Agents Problem.....	65
6.2.1	Case I: Obstacles: Fixed; Goal: Fixed.....	65
6.2.2	Case II: Obstacles: Fixed & Moving (Both); Goal: Fixed.....	70
6.2.3	Case III: Obstacles: Fixed; Goal: Moving.....	73
6.2.4	Case IV: Obstacles: Fixed & Moving (Both); Goal: Moving....	78
6.3	Four Agents Problem.....	80
6.3.1	Case I: Obstacles: Fixed; Goal: Fixed.....	80
6.3.2	Case II: Obstacles: Fixed & Moving (Both); Goal: Fixed.....	86
6.3.3	Case III: Obstacles: Fixed; Goal: Moving.....	88
6.3.4	Case IV: Obstacles: Fixed & Moving (Both); Goal: Moving....	93
	<b>CHAPTER 7. CONCLUSIONS &amp; FUTURE SCOPE.....</b>	<b>96</b>
7.1	Main Conclusions.....	96
7.2	Future Scope of Work.....	98
	<b>REFERENCES.....</b>	<b>99</b>

## LIST OF FIGURES

Figure 3.1 Agent - Environment interaction in the Reinforcement Learning.....	19
Figure 4.1 A sample grid world with green block as obstacles, red block as goal and green block as starting block .....	34
Figure 5.1 Environment in which SARSA is effective.....	47
Figure 6.1.1 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case I(a) .....	52
Figure 6.1.2a Path traced for case I(a) by single agent for Q-learning .....	52
Figure 6.1.2b Path traced for case I(a) by single agent for SARSA .....	52
Figure 6.1.2c Path traced for case I(a) by single agent for Phe-Q .....	52
Figure 6.1.2d Path traced for case I(a) by single agent for Phe-SARSA.....	52
Figure 6.1.3 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case I(b) .....	53
Figure 6.1.4a Path traced for case I(b) by single agent for Q-learning .....	53
Figure 6.1.4b Path traced for case I(b) by single agent for SARSA .....	53
Figure 6.1.4c Path traced for case I(b) by single agent for Phe-Q .....	54
Figure 6.1.4d Path traced for case I(b) by single agent for Phe-SARSA.....	54
Figure 6.1.5 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case I(c) .....	54
Figure 6.1.6a Path traced for case I(c) by single agent for Q-learning .....	55
Figure 6.1.6b Path traced for case I(c) by single agent for SARSA .....	55
Figure 6.1.6c Path traced for case I(c) by single agent for Phe-Q .....	55
Figure 6.1.6d Path traced for case I(c) by single agent for Phe-SARSA.....	55
Figure 6.1.7 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case II(a) .....	56
Figure 6.1.8a Path traced for case II(a) by single agent for Q-learning .....	56
Figure 6.1.8b Path traced for case II(a) by single agent for SARSA .....	56
Figure 6.1.9 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case II(b) .....	57
Figure 6.1.10a Path traced for case II(b) by single agent for Q-learning .....	57
Figure 6.1.10b Path traced for case II(b) by single agent for SARSA .....	57



Figure 6.1.11 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case III(a) .....	58
Figure 6.1.12a Path traced for case III(a) by single agent for Q-learning .....	59
Figure 6.1.12b Path traced for case III(a) by single agent for SARSA .....	59
Figure 6.1.12c Path traced for case III(a) by single agent for Phe-Q .....	59
Figure 6.1.12d Path traced for case III(a) by single agent for Phe-SARSA.....	59
Figure 6.1.13 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case III(b) .....	60
Figure 6.1.14a Path traced for case III(b) by single agent for Q-learning .....	60
Figure 6.1.14b Path traced for case III(b) by single agent for SARSA .....	60
Figure 6.1.14c Path traced for case III(b) by single agent for Phe-Q .....	60
Figure 6.1.14d Path traced for case III(b) by single agent for Phe-SARSA.....	60
Figure 6.1.15 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case III(c) .....	61
Figure 6.1.16a Path traced for case III(c) by single agent for Q-learning .....	62
Figure 6.1.16b Path traced for case III(c) by single agent for SARSA .....	62
Figure 6.1.16c Path traced for case III(c) by single agent for Phe-Q .....	62
Figure 6.1.16d Path traced for case III(c) by single agent for Phe-SARSA.....	62
Figure 6.1.17 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case IV(a).....	63
Figure 6.1.18 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case IV(b).....	63
Figure 6.2.1 Plot between no. of steps required to reach the goal and no. of attempts for 2 agents; case I(a) .....	65
Figure 6.2.2a Path traced for case I(a) by two agents for Q-learning .....	66
Figure 6.2.2b Path traced for case I(a) by two agents for SARSA .....	66
Figure 6.2.2c Path traced for case I(a) by two agents for Phe-Q .....	66
Figure 6.2.2d Path traced for case I(a) by two agents for Phe-SARSA.....	66
Figure 6.2.2e Path traced for case I(a) by two agents for Q-Swarm....	66
Figure 6.2.2f Path traced for case I(a) by two agents for SARSA-Swarm.....	66
Figure 6.2.3 Plot between no. of steps required to reach the goal and no. of attempts	

for 2 agents; case I(b) .....	67
Figure 6.2.4a Path traced for case I(b) by two agents for Q-learning .....	68
Figure 6.2.4b Path traced for case I(b) by two agents for SARSA .....	68
Figure 6.2.4c Path traced for case I(b) by two agents for Phe-Q .....	68
Figure 6.2.4d Path traced for case I(b) by two agents for Phe-SARSA.....	68
Figure 6.2.4e Path traced for case I(b) by two agents for Q-Swarm... ..	68
Figure 6.2.4f Path traced for case I(b) by two agents for SARSA-Swarm.....	68
Figure 6.2.5 Plot between no. of steps required to reach the goal and no. of attempts for 2 agents; case I(c) .....	69
Figure 6.2.6a Path traced for case I(c) by two agents for Q-learning .....	69
Figure 6.2.6b Path traced for case I(c) by two agents for SARSA .....	69
Figure 6.2.6c Path traced for case I(c) by two agents for Phe-Q .....	69
Figure 6.2.6d Path traced for case I(c) by two agents for Phe-SARSA.....	69
Figure 6.2.6e Path traced for case I(c) by two agents for Q-Swarm... ..	70
Figure 6.2.6f Path traced for case I(c) by two agents for SARSA-Swarm.....	70
Figure 6.2.7 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case II(a).....	70
Figure 6.2.8a Path traced for case II(a) by two agents for Q-learning .....	71
Figure 6.2.8b Path traced for case II(a) by two agents for SARSA .....	71
Figure 6.2.9 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case II(b).....	72
Figure 6.2.10a Path traced for case II(b) by two agents for Q-learning .....	72
Figure 6.2.10b Path traced for case II(b) by two agents for SASA .....	72
Figure 6.2.11 Plot between no. of steps required to reach the goal and no. of attempts for 2 agents; case III(a) .....	73
Figure 6.2.12a Path traced for case III(a) by two agents for Q-learning .....	73
Figure 6.2.12b Path traced for case III(a) by two agents for SARSA .....	74
Figure 6.2.12c Path traced for case III(a) by two agents for Phe-Q .....	74
Figure 6.2.12d Path traced for case III(a) by two agents for Phe-SARSA.....	74
Figure 6.2.12e Path traced for case III(a) by two agents for Q-Swarm.....	74
Figure 6.2.12f Path traced for case III(a) by two agents for SARSA-Swarm.....	74

Figure 6.2.13 Plot between no. of steps required to reach the goal and no. of attempts for 2 agents; case III(b) .....	75
Figure 6.2.14a Path traced for case III(b) by two agents for Q-learning .....	75
Figure 6.2.14b Path traced for case III(b) by two agents for SARSA .....	76
Figure 6.2.14c Path traced for case III(b) by two agents for Phe-Q .....	76
Figure 6.2.14d Path traced for case III(b) by two agents for Phe-SARSA.....	76
Figure 6.2.14e Path traced for case III(b) by two agents for Q-Swarm.....	76
Figure 6.2.14f Path traced for case III(b) by two agents for SARSA-Swarm.....	76
Figure 6.2.15 Plot between no. of steps required to reach the goal and no. of attempts for 2 agents; case III(c) .....	77
Figure 6.2.16a Path traced for case III(c) by two agents for Q-learning .....	77
Figure 6.2.16b Path traced for case III(c) by two agents for SARSA .....	77
Figure 6.2.16c Path traced for case III(c) by two agents for Phe-Q .....	77
Figure 6.2.16d Path traced for case III(c) by two agents for Phe-SARSA.....	77
Figure 6.2.16e Path traced for case III(c) by two agents for Q-Swarm.....	78
Figure 6.2.16f Path traced for case III(c) by two agents for SARSA-Swarm.....	78
Figure 6.2.17 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case IV(a).....	79
Figure 6.2.18 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case IV(b).....	79
6.3.1 Plot between no. of steps required to reach the goal and no. of attempts for 4 agents; case I(a) .....	81
Figure 6.3.2a Path traced for case I(a) by four agents for Q-learning .....	81
Figure 6.3.2b Path traced for case I(a) by four agents for SARSA .....	81
Figure 6.3.2c Path traced for case I(a) by four agents for Phe-Q .....	82
Figure 6.3.2d Path traced for case I(a) by four agents for Phe-SARSA.....	82
Figure 6.3.2e Path traced for case I(a) by four agents for Q-Swarm... ..	82
Figure 6.3.2f Path traced for case I(a) by four agents for SARSA-Swarm.....	82
6.3.3 Plot between no. of steps required to reach the goal and no. of attempts for 4 agents; case I(b) .....	83
Figure 6.3.4a Path traced for case I(b) by four agents for Q-learning .....	83

Figure 6.3.4b Path traced for case I(b) by four agents for SARSA .....	83
Figure 6.3.4c Path traced for case I(b) by four agents for Phe-Q .....	84
Figure 6.3.4d Path traced for case I(b) by four agents for Phe-SARSA.....	84
Figure 6.3.4e Path traced for case I(b) by four agents for Q-Swarm....	84
Figure 6.3.4f Path traced for case I(b) by four agents for SARSA-Swarm.....	84
6.3.5 Plot between no. of steps required to reach the goal and no. of attempts for 4 agents; case I(c) .....	85
Figure 6.3.6a Path traced for case I(c) by four agents for Q-learning .....	85
Figure 6.3.6b Path traced for case I(c) by four agents for SARSA .....	85
Figure 6.3.6c Path traced for case I(c) by four agents for Phe-Q .....	85
Figure 6.3.6d Path traced for case I(c) by four agents for Phe-SARSA.....	85
Figure 6.3.6e Path traced for case I(c) by four agents for Q-Swarm....	86
Figure 6.3.6f Path traced for case I(c) by four agents for SARSA-Swarm.....	86
Figure 6.3.7 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case II(a).....	86
Figure 6.3.8 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case II(b).....	87
Figure 6.3.9 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case III(a).....	88
Figure 6.3.10a Path traced for case III(a) by four agents for Phe-Q .....	89
Figure 6.3.10b Path traced for case III(a) by four agents for Phe-SARSA.....	89
Figure 6.3.10c Path traced for case III(a) by four agents for Q-Swarm....	89
Figure 6.3.10d Path traced for case III(a) by four agents for SARSA-Swarm.....	89
Figure 6.3.11 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case III(b).....	90
Figure 6.3.12a Path traced for case III(b) by four agents for Phe-Q .....	90
Figure 6.3.12b Path traced for case III(b) by four agents for Phe-SARSA.....	90
Figure 6.3.12c Path traced for case III(b) by four agents for Q-Swarm....	91
Figure 6.3.12d Path traced for case III(b) by four agents for SARSA-Swarm.....	91
Figure 6.3.13 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case III(c).....	91

Figure 6.3.14a Path traced for case III(c) by four agents for Phe-Q .....	92
Figure 6.3.14b Path traced for case III(c) by four agents for Phe-SARSA.....	92
Figure 6.3.14c Path traced for case III(c) by four agents for Q-Swarm... ..	92
Figure 6.3.14d Path traced for case III(c) by four agents for SARSA-Swarm.....	92
Figure 6.3.15 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case IV(a)	93
Figure 6.3.16 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case IV(b)	94

## **LIST OF TABLES**

Table 6.1 Computational Time for the Single Agent Cases for the four algorithms simulated.....	64
Table 6.2 Computational Time for the Two Agents Cases for the six algorithms simulated.....	80
Table 6.3 Computational Time for the Four Agents Cases for the six algorithms simulated.....	95

# CHAPTER 1

## INTRODUCTION

This chapter presents the motivation behind the work done. It aims at implementation of the Optimization algorithms in Reinforcement Learning (RL) methods and has also been provided with the thesis organization.

### 1.1 OVERVIEW

Machine learning for robots or mechanisms of all kinds has been a great challenge for engineers and scientists, from the beginning days of the computers. The learning characteristic of animals in the simplest search jobs, such as avoiding obstacles while doing some task or searching for food, turns out to be extremely difficult to reproduce in artificial mechanical devices, real or simulated. This thesis shows that how reinforcement learning with the help of nature inspired algorithms can help to solve such problems.

“Reinforcement Learning”, this word is new for human concept and can be traced back to the stone age. Humans learned long ago that we learn from our mistakes and that we should learn if we want to improve over time. Learned lessons can be passed from one generation to the other by changing the way we think, interact or work. Over the last five decades, it has been shown that machines can be made to learn similar to humans. “Machine Learning” is used to define many different applications from a military drone to a robotic carpet cleaner. It includes different types of learning like Supervised or Unsupervised Learning. In this thesis we will look into Reinforcement Learning algorithms and their hybrid forms.

Reinforcement learning is neither supervised nor non-supervised kind of learning but forms a third category of learning. The evolution of Reinforcement Learning from its beginning in the 1950s to the present day has been very impressive. In the last two decades, faster computers with more memory led to the implementation of learning algorithms like the single agent Q-Learning algorithm by Watkins [4]. This Q-Learning

algorithm was a major breakthrough in Reinforcement Learning and it later on became the foundation of many algorithms.

## **1.2 REINFORCEMENT LEARNING**

Machine learning is programming to optimize a performance criterion using example data or previous observations. Learning a model with partially defined parameters is the execution of a computer program to optimize the parameters of the model using the training data or previous observations. Machine learning uses the theory of statistics in building mathematical models, because the main task is making inference from a sample. In applications such as navigation, grabbing, and exploration, the output of the system is a sequence of actions. In such a case, a single action is not important; what is important is the policy that defines the sequence of correct actions to reach the goal given the current state of the environment. Such learning methods are called reinforcement learning (RL) algorithms[16]. In RL, the learner is a decision-making agent that takes actions in an environment and receives reward (or penalty) for its actions in trying to solve a problem. After a set of trial-and error runs, it should learn the best policy, which is the sequence of actions that maximizes the total reward[16]. One of the most famous methods of completing tasks in robotics is the use of behavior based models [15]. Each behavior requires a sequential set of actions to be completed and RL is the best candidate for such systems.

## **1.3 PATH SEARCH AND RL**

In many real world problems, the need of automation is arising. With the world shifting form a place where only living beings existed to a place where artificial living-like things also exist, it is becoming very crucial to use robots to ease our day to day work. Many works has been done in machine learning where a robot is made to mimic human actions and quite some achievement has also been made. One such human-like action for robot could be to navigate in a space or to search a space to find some particular goal, which could be food or other robots or other living beings or any destination. To navigate or move in an unknown environment, whose model is not known might be difficult task for robot.



The robot brain organizes a vocabulary of keywords that describe the robot's perception of the environment. The results of its experiences are processed by a model that finds cause and effect relationships between executed actions and changes in the environment. This allows the robot to learn from the consequences of its actions in the real world. More specific, the robot starts with a training procedure. The basic idea of RL is to tell a robotic agent when it is behaving good or bad and make it derive a suitable behavior from these reinforcement signals. Recently, RL has begun being used on simulated and real robots. In the spirit of embodied cognitive science, the investigations will include experiments on a real robot. An understanding has emerged from the findings in science that it is not feasible to separately investigate the mind and body of humans, animal, or robots when the goal is to gain knowledge about intelligent behavior. These things are interconnected and have to be treated as a whole.

#### **1.4 ANT COLONY OPTIMIZATION (ACO)**

This algorithm is a member of the ant colony algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. Initially proposed by Marco Dorigo in 1992 in his PhD thesis[5], the first algorithm was aiming to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants. In the natural world, ants wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food.

Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths

chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained.

Thus, when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to all the ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve.

## **1.5 PARTICLE SWARM OPTIMIZATION (PSO) AND GROUP EFFORT**

Particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate (particles) solutions, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position but, is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

PSO is originally attributed to Kennedy, Eberhart and Shi [11, 13] and was first intended for simulating social behaviour[14], as a stylized representation of the movement of organisms in a bird flock or fish school. The algorithm was simplified and it was observed to be performing optimization. PSO is a metaheuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as PSO do not guarantee an optimal solution is ever found. PSO can therefore also be used on optimization problems that are partially irregular, noisy, change over time, etc.

## **1.6 OBJECTIVE OF WORK**

As in automation problems the application of robotics is increasing day by day. With the advancement in the technology, a shift is being seen towards implementation

of bio-inspired algorithms from the conventional methods or algorithms. Thus, in the areas of the robotics problem also, more bio-inspired algorithms are being used.

In any search or path planning problem, it becomes essential to optimally reach the solution in less time. By using reinforcement learning methods we can achieve this. To implement this for a multi agent dynamic situation, the conventional methods do not follow and there is a need to switch from the conventional methods to the hybrid ones. Thus, reinforcement learning has been combined with the bio-inspired algorithms like PSO and ACO to reach the optimal solution in minimum time.

## **1.7 ORGANIZATION OF THESIS**

CHAPTER – 2 This chapter consists of the literature survey on various reinforcement learning algorithms and the different evolutionary methods that have been implemented with it. The major emphasis is given on the PSO and ACO which have been implemented with RL methods of Q-learning and SARSA.

CHAPTER – 3 This chapter deals with the description of the concept of RL, Markov Decision Process (MDPs) and Temporal Difference (TD) method which is used to solve the problems of RL. A short description about the two most commonly used TD(0) algorithms, i.e. Q-learning and SARSA is also given.

CHAPTER – 4 This chapter defines the Grid World problem and various Multi-Agent Reinforcement Learning have been stated. The Friend and Foe Q-Learning algorithm has been discussed in detail.

CHAPTER – 5 Three evolutionary algorithms used with RL have been described in this chapter: Q-Swarm, SARSA-Swarm, Phe-Q and a new algorithm has been proposed SARSA-Q.

CHAPTER – 6 This chapter presents results and discussion.

CHAPTER – 7 In this chapter, the main conclusions have been drawn out and some future work related to the research have been suggested.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 GENERAL**

This chapter consists of literature survey on different algorithms in this project. Various books and papers related to reinforcement learning algorithms, evolutionary algorithms: Particle Swarm Optimization and Ant Colony Optimization have been studied.

#### **2.2 LITERATURE REVIEW**

M. C. Cammaerts-Tricot and J. C. Verhaeghe [1] analysed the trail pheromone production and trail following behaviour of workers of *Myrmica rubra* of different age groups, categorized by their cuticular pigmentation. The dimensions of the poison gland reservoir increase as workers grow older. The capabilities of ants help the colony for recruitment to repel an enemy or to exploit a source of food occurs in its foraging area.

D. P. Bertekas et al. [2] gave an elaborated description on the stochastic optimal control methods. They implemented the various control strategies in the discrete time space.

S. Goss et al. [3] showed various methods for the self organization in the Argentinean ant colony. These various methods were implemented for finding the shortest path in a maze.

Watkins [4] showed that Q-learning is a simple way for agents to learn how to act optimally in controlled Markovian domains. It amounts to an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states.

Dorigo [5] researched on a new metaheuristic for optimization which was often initially focused on proof-of-concept applications. He provided a survey on theoretical results on ant colony optimization. Some research efforts were directed at gaining a deeper understanding of the behavior of ant colony optimization algorithms.

Littman [6] showed minimax criterion allows the agent to converge to a fixed policy that is guaranteed to converge. This is certainly true to some extent but any such agent will in principle be vulnerable to a devious form of trickery in which the opponent leads the agent to learn a poor policy and then exploits it. He also showed that RL can be used in multi-agent scenarios and adversarial environments are well behaved as in that optimality is guaranteed against some random opponent. In such an environment the multi-agent RL are less behaved, but strong assumption needs to be made out about other agents to guarantee convergence.

Putterman [7] described about the infinite-horizon discrete-time models with discrete state spaces and described the Markov Decision Processes (MDPs). He also described about the modified policy iteration, multi-chain simulations with sensitive optimality and average reward criterion.

Rummery and Niranjan [8] compared the performance of different RL algorithms on a realistic robot navigation problem, where a simulated mobile robot is trained to guide itself to a goal position in the presence of obstacles. They showed that on-line learning algorithms are less sensitive to the choice of training parameters than backward replay.

L. R. Leerink [9] applied ant trail formation and foraging methods to the problem of exploration in a discrete environment with delayed reinforcement. The exploration strategy used was the various mechanisms that are found in ant trail formation in the adaptive heuristic critic framework, and was applied to a robot navigation task. Simulations indicate that in terms of efficiency the mechanisms used by a single ant perform better than undirected exploration methods, but not as well as specialized directed algorithms. However, when multiple robots simultaneously explore the environment the performance increases in a superlinear manner, resulting in an emergent collective ability larger than that possessed by the individual robots.

B. Holldobler et al. [10] explained the various categories in which the ant system is divided and how each ant category work together to reach the goal. He showed ant colony as intricate super organism in which individual ants are only small,

indispensable fairly mechanical, easily replaceable walking batteries of exocrine glands that sense their world primarily through the chemical secreted by them known as pheromone.

J. Kennedy et al. [11, 13, 14] first intended for simulating social behaviour as a stylized representation of the movement of organisms in a bird flock or fish school. They introduced the Particle Swarm Optimization algorithm and later it was simplified and observed to be performing optimization.

Bertekas et al. [12] explained the concept of dynamic programming and showed that how it can be solved and implemented with the help of neural networks.

R. C. Arkin [15] showed that the most famous methods of completing tasks in robotics is the use of behavior based models. And that each behavior required a sequential set of actions to be completed.

R. S. Sutton and A. G. Barto [16] gave the concept of Reinforcement Learning and explained the various terminologies related to the reinforcement learning. They showed that temporal difference method is used to solve the reinforcement learning problem and explained the optimality of TD(0) method. Various examples are shown by the authors to relate the concept of reward and the value functions.

Junling Hu and M. P. Wellman [17, 27] experimented on the general-sum stochastic games and solved the problem using Q-learning. They also solved this problem for the Nash Q-learning. Nash Q-learning gave better results as compared to Q-learning because of the Nash factor taken into account which uses collective reward function rather than individual rewards.

Singh et al. [18] examined the convergence of single-step on-policy RL algorithms for control. They showed that On-policy algorithms cannot separate exploration from learning and therefore must confront the exploration problem directly and hence proved that convergence results for several related on-policy algorithms with both decaying exploration and persistent exploration.

Vaughan et al. [19, 25] implemented ant-like self-organizing behaviour to coordinate a multi-robot system where the robots transport objects between various

locations. Robots used here had shared memory to communicate path information rather than physically laying trail of synthetic pheromone.

M. L. Littman [20, 21] described a set of reinforcement-learning algorithms based on estimating value functions and presented convergence theorems for these algorithms. They analysed and proved the convergence for Q-learning, Minmax-Q learning, Nash Q-learning and Team Q-learning.

In Parunak V. D., et al. [22, 23, 24] the pheromone trails were used to construct potential fields. Unmanned vehicles were used to navigate, directed by the potential gradients. The ant system had different ‘flavours’ of pheromone. Each pheromone kind was exclusive in a way that it was associated with a peculiar feature of the environment and has their own evaporation and diffusion rates resulting in different dynamics.

M. Monekosso and P. Remagnino [26, 28] first introduced the Phe -Q algorithm which has the similar structure for rule updation as Q-learning with an addition term in the update equation called the belief factor. It is a function of the pheromone level in each cell and is associated with the state-action pair. It was implemented for grid world with fixed obstacles.

Hitoshi Jimal and Yasuaki Kuroe [29, 30] proposed a hybrid algorithm called Swarm-Q for multi-agent environment in which each agent learns individually using parallel Q-learning and also learns using interaction using PSO utilizing personal best and global best Q-values. Later, they also used the same PSO based Q updating rule with SARSA as the basic learning method and showed that it is more effective for an environment which has a large negative reward. The algorithm optimizes quickly than the normal Q-learning and SARSA and also Swarm-Q.

L. Busoniu et al. [31, 34] discussed about the various Multi-Agent Reinforcement Learning (MARL) techniques for fully cooperative, fully competitive and mixed problems. They pointed out the main benefits and challenges of the various MARL algorithms. They also described the deterministic and stochastic MDPs and characterized their optimal solutions. They explained the concept of reinforcement learning and the dynamic programming. They explained about the Q-learning and SARSA and explained the

need of exploration over exploitation. Various applications such as speed control of a DC motor were solved using Q- learning and SARSA.

Chia-Feng Juang et al. [32] proposed the design of a fuzzy controller by Ant Colony Optimization (ACO) incorporated with Fuzzy-Q Learning, called ACO-FQ, with reinforcements. For a fuzzy controller, a list of all candidate consequent control actions of each fuzzy rule were made. Each candidate in the consequent part of a rule is assigned with a corresponding Q-value. Searching for the best one among all combinations is partially based on pheromone trail and partially based on Q-values. Results were verified for a water bath temperature control system.

Kadlecek D. and Nahodil P. [33] integrated rigorous methods of reinforcement learning and control engineering with a behavioral approach to the agent technology. The main outcome is a hybrid architecture for intelligent autonomous agents targeted to the Artificial Life like environments. Learning and control was realized by multiple RL controllers working in a hierarchy of Semi Markov Decision Processes (SMDP). Used model free  $Q(\lambda)$  learning works online, the agents gain experiences during interaction with the environment.

Wei Wu et al. [35] presented a control method based on multi-agent for traffic signals. Reinforcement learning algorithm was used to optimize traffic flow in the intersection. The genetic algorithm intended to introduce a global optimization criterion to each of the local learning processes that optimize the cycle of traffic signals and green-ratio. Areawide coordination was achieved by game theory. Here, local optimization with global optimization to optimize traffic signal in multi-intersection. Simulation results indicate that our presented method is superior than traditional control one.

J. Pazis et al. [36] presented a novel, computationally-efficient method, called Adaptive Action Modification, for realizing continuous-action policies, using binary decisions corresponding to adaptive increment or decrement changes in the values of the continuous action variables. They proposed an approach which approximates any continuous action space to arbitrary resolution and can be combined with any discrete-action reinforcement learning algorithm for learning continuous-action policies. They coupled Q-



Learning, Fitted Q-Iteration, and Least-Squares Policy Iteration and implemented it on the continuous state-action Inverted Pendulum and Bicycle Balancing and Riding domains.

Shu Da Wang et al. [37] constructed a multi-agent simulation system based on reinforcement learning algorithms, achieve real-time simulation of multi-agent, and multi-agent to get effect quickly, and to quickly construct surrounded conduct by mobile groups, the conduct of the system to achieve the global optimum effect. Seige type group problem was taken here. Two groups of agents were simulated to compete and reach the goal using Q-learning.

A. T. Evangolelos et al. [38] used reinforcement learning to find path in an environment. They used integral control to find path in any general environment.

J. Glascher et al. [39] used RL sequential experience with situations ("states") and outcomes to assess actions. Using functional magnetic resonance imaging in humans solving a probabilistic Markov decision task, they found the neural signature of an SPE in the intraparietal sulcus and lateral prefrontal cortex. Their finding supports the existence of two unique forms of learning signal in humans, which may form the basis of distinct computational strategies for guiding behavior.

Qiangfeng P. L. et al. [40] presented a distributed reinforcement learning system that leverages on expert coordination knowledge to improve learning in multi-agent problems. Scenario was taken where agents can communicate with their neighbors but this communication structure and the number of agents was changed over time. Experiment results were carried out for a tactical realtime strategy and soccer games.

Romero F. T. et al. [41] introduced a mobile robotic system to learn through reinforcement, which allows it to navigate within a dynamic environment avoiding any obstacle it might encounter. The learning system was implemented with two neural networks. Both neural networks use reinforcement learning by means of the Hebb rule. In this paper, it was shown that there may be a case when the robot is stuck in a region with such a configuration that directly affects it and prevents it from navigating the entire environment.

Seiichi A. and Takao M. [42] proposed a new framework for combinatorial auctions with Q-learning agent. They showed how an intelligent agent learns in combinatorial auctions. They applied a framework of machine learning to combinatorial auctions to extract intelligence about bidding behavior. It was shown that the agent obtains strategies for behavior by considering combinatorial auctions as outside environment. Here, Q-learning approach was useful to obtain knowledge for winner.

M. Stocia et al. [43] used reinforcement learning method for the industrial robot problem. They took the problem of navigation in which the robot needed to transfer objects from point to point and implemented Q-learning algorithm for their learning.

Ji-Hwan Son et al. [44, 59] demonstrated movement control of the insect and enhanced control of the robot through its own learning progress via reinforcement learning. It was shown that insect occasionally exhibited uncertain and complex behavior and that interaction mechanism was affected by weather and other unknown properties of a real environment, resulting in more complex behaviors. To solve this, they proposed fuzzy logic-based cooperative RL for sharing knowledge among agents. They designed a fuzzy logic-based expertise measurement system for cooperative RL. The structure makes artificial robots share knowledge under measuring performance evaluation of each agent.

Devin G. et al. [45] et al. applied the Partially-Observable Markov Decision Processes (POMDPs) to a robotic navigation task under state and sensing uncertainty. This method provided a useful action model that gave a policy with similar overall expected reward compared to the true action model with significant computational savings. It was shown that this technique of building problem-dependent approximations can provide significant computational advantages and can help expand the complexity.

S. Zhiguo et al. [46] gave an improved Q-learning algorithm based on pheromone mechanism. They implemented it for a swarm of four robots to find path in a maze. The algorithm used two stages learning in which individual robots learned using RL and the pheromone level and the overall learning was done using the pheromone levels.

Chun-Tse Lin et al. [47] solved the path tracking problem of a prototype walking-aid robot which features the human-robot interactive navigation. A practical fuzzy

controller was proposed for the path tracking control under reinforcement learning ability. The inputs taken for the design of fuzzy controller were, the error distance and the error angle between the current and the desired position and orientation, respectively. The controller outputs taken was the voltages applied to the left- and right-wheel motors. A heuristic fuzzy control with the Sugeno-type rules was designed based on a model-free approach. The fuzzy control rule was designed with the aid of Q-learning approach.

Mohammed I. A. et al. [48] gave a study of various class of multi-agent graphical games denoted by differential graphical games, where interactions between agents are prescribed by a communication graph structure. Nash solutions were given in terms of solutions to a set of coupled continuous-time Hamilton-Jacobi Bellman equations. An online multi-agent method based on policy iterations was developed using a critic network to solve all the Hamilton-Jacobi-Bellman equations simultaneously for the graphical game. Here, an online adaptive Integral Reinforcement Learning structure using critic structures was used to solve the differential graphical game.

O. Krigolson et al. [49] gave an analogy of the reinforcement learning and the way humans learn from the errors. They also explained how we make decision based on our reinforcement learning mechanism.

Figuroa R. et al. [50] demonstrated a novel solution to the inverted pendulum problem extended to UAVs, specifically quadrotors. The solution is provided by reinforcement learning (RL) to generate a control policy to balance the pendulum using Continuous Action Fitted Value Iteration (CAFVI) which is a RL algorithm for highdimensional input-spaces. This technique combined learning of both state and state-action value functions in an approximate value iteration setting with continuous inputs.

J. S. Campbell et al. [51] used the delayed reinforcement learning method in a single agent problem. They implemented various types of models in the Q-learning for the stochastic reinforcement using delays.

Bashan Z. et al. [52] developed a navigation technology based on the Q-learning algorithm. Here, an autonomous mobile robot was required to navigate in an unknown maze

and move out of it as soon as possible. They showed this technique was effective and successful to help a robot navigate in an unknown environment and avoid obstacles.

Huan T. et al. [53] proposed a novel evolutionary reinforcement learning method and applied it to robotic imitation learning, which integrates EDA and PI2 learning algorithm. This algorithm provides a solution to integrate exploratory learning methods with traditional reinforcement learning algorithms. This work can also be applied in other domains where the problems to be solved could be described as a well-known nonlinear state system.

Yunfei Z. et al. [54] developed a hierarchical controller to avoid randomly moving obstacles in autonomous navigation of a robot. The developed method consisted of two parts: a highlevel Q-learning controller for choosing an optimal plan for navigation and a low-level, appearance-based visual servo (ABVS) controller for motion execution.

Vasquez D. et al. [55] compared various IRL based learning methods and feature sets for socially compliant robot navigation in crowds. They provided three important insights a) the importance of the default cost feature; b) the need of motion prediction to obtain smoother human-like motion; and c) for i.e. linear combination of weights cost, it seems to be better to put the effort on feature design than on the learning algorithms. Conversely, in order to simplify the task of designing features, richer, more complex cost functions and learning algorithms are required.

Bischoff B. et al. [56] investigated model-based reinforcement learning in particular the probabilistic inference for learning control method (PILCO), with the case of sparse data to speed up learning. This approach was evaluated in simulation as well as on a physical robot. They showed that by including prior knowledge, policy learning can be sped up in presence of sparse data.

Chao Yu et al. [57] proposed a multi agent learning approach to solve coordination problems by exploiting agent independence in loosely coupled multi agent systems. This approach enabled agents to learn an efficient coordinated policy through dynamic adaptation of the estimation of agent independence. This method required neither

prior knowledge about the structure of the domain nor assumptions about the learning agents.

H. Modares et al. [58] used reinforcement learning for the robot movements. They optimized the the steps taken by the robots to peform human-like activities using reinforcement learning. Q-learning was used as the method for learning of the robot.

## **CHAPTER 3**

### **REINFORCEMENT LEARNING**

#### **3.1 BIOLOGICAL INSPIRATION**

The basic idea that we learn from our environment by interacting is probably the foremost one to occur to us when we think about the process of learning. When a child plays, wave arms, or gets injured, it does not have an explicit teacher, though it has a certain direct sensor-motor link to its environment. Using this connection, a vast repository of information about cause and effect, about the results of actions, and about what should be done in order to achieve target. Throughout our existence, experiences are undoubtedly a major source of knowledge about our environment and us. Learning from interactions is the initial idea behind almost all theories of intelligence and learning. “Reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning problem”. [16]

#### **3.2 INTRODUCTION**

“Reinforcement Learning is learning what to do, how to map situations to actions, so as to maximize a numerical reward signal.”[16] In RL, a controller interacts with a process, by means of three signals: an action signal, which allows the controller to influence the process, a state signal, describing the state of the process, an action signal, which influences the process, and a scalar reward signal, providing the controller with feedback on its immediate performance.

The concept of rewarding for a particular set of actions is not a new concept in our society. The reward generated is an evaluation of the quality of transition between previous state and new state. This can be related to our daily lives: A person will be more inclined to do a task if there is a positive reward for executing it. Of course, this cannot characterise all of human behaviour, but we can see how RL influences our life.

To understand Reinforcement Learning, a simple case of an agent in an environment can be taken. In a single agent situation, the agent interacts with the environment and determines the actions that will earn the maximum rewards. For example,

inputs are provided by the environment to the agent and the agent then interacts with the environment with different outputs in form of actions. This is the main difference between RL and other learning methods. A reward is also given by the environment to the agent for each of the actions. The agent then learns that few actions gives better rewards than others and it thus learn to reproduce these actions to maximize its future rewards.

The RL framework has been used to solve various optimization processes and have been applied to many varied applications, e.g., automatic control, robot navigation, operations research, artificial intelligence, economics, robot navigation [35, 44, 46, 57, 59].

### **3.3 ELEMENTS & BASIC RL FRAMEWORK**

Other main sub-elements apart from state-action can be identified in the RL system are: a policy, a reward function and a value function.

#### **3.3.1 Policy**

A policy is defined as the learning agent's behaviour. It is a mapping from perceived states of the environment to actions to be taken when in those states. This could be related to a set of stimulus-response rules or associations in psychology. For countable states, policy is generally a look up table or simple function, while for uncountable or continuous state spaces it involves an extensive computation such as a search process. It is the vital part of an RL agent in the sense that it alone is sufficient to determine the agent behaviour.

#### **3.3.2 Reward Function**

The goal in an RL problem is defined mainly by the reward function. It maps each perceived state of the environment to a scalar value, a reward, which indicates the inherent desirability of that state. The sole objective of an RL agent is to maximize the total reward received by it in the long run.

The reward function defines good and bad episodes for the agent. In a biological system, reward can be identified as liking and pain. They are direct and defining features of the problem which is faced by the agent. Hence, the reward function is never

altered by the agent. However, it may serve as a basis for policy alteration. As an example, if an action as selected by the policy is followed by less reward, then this policy might be changed to select some other action in that situation in the future.

### 3.3.3 Value Function

A reward function is a measure for immediate action and a value function is for the long run. Value for a state could be defined as the total amount of reward that an agent can expect to collect over the future, initiating from that state. Whereas rewards tells about the instant, intrinsic desirability of environmental states, values indicate the continuing desirability of states after considering the states that are likely to be followed, and the rewards then available in those states. As an example, particular state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards or vice-versa. In terms of humans, rewards are like preferences (if high) and discomfort (if low), whereas values are more refined and farsighted judgment of how pleased or displeased we are that our environment is in a particular state.

### 3.3.4 Basic Framework of RL

Reinforcement Learning, also known as enhanced learning, is a machine learning method which optimizes the result by goal-oriented learning which study by direct interaction with the environment. In Supervised learning method the training information required is instructional whereas in the reinforcement learning, training information required is evaluative and provides an important intelligent control method for the agent. The main purpose of reinforcement is studying the optimal mapping from state to action, so as to maximum the reward signal. [49]

Figure 3.1 from [16] illustrates clearly the different interactions between the agent and the environment. Both the agent and the environment interact at finite time steps  $k = 0; 1; 2; 3; \dots$  [16] This means that each interaction will be done at a predetermined time step. The environment provides the agent with the state  $s_k$  element of  $S$ , where  $S$  is the set of possible states [16]. The agent is able to choose an action  $a_k$  at element of  $A(s_k)$ , where  $A(s_k)$  is the set of possible actions in state  $s_k$  [16]. For time step  $k + 1$ , the



environment will provide a reward,  $r_{k+1}$ , which is the reward function  $R$  and a new state  $s_{k+1}$  to the agent [16]. This reward is due to its action in the previous state  $s_k$  and the transition to the new state  $s_{k+1}$ .

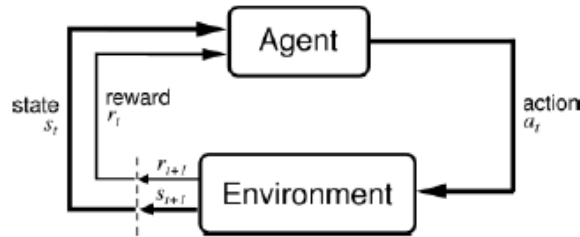


Figure 3.1 Agent - Environment interaction in the Reinforcement Learning

This is where RL comes into play. Each reward is associated with different actions and develop strategies that are called policies. The policy,  $\pi_k(s; a)$ , can be defined as the probability that  $a_k = a$  if  $s_k = s$ , where the  $k$  represents the next step [16]. The agent thus has to associate different probabilities to each action to maximize its rewards. All of the RL research starts with this simple concept and develops different methods of using the reward function. Expected return of an agent is linked to the environmental reward function. The reward function can be very different from one environment to the other.

For example, it describes which actions or series of actions will provide what reward. As per discounted reward, the expected reward diminishes over time. We can illustrate this by the equation of the expected discounted return:

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \dots + \gamma^n r_{n+k+1} \quad (3.1)$$

where,  $\gamma$  is a parameter,  $0 \leq \gamma \leq 1$ , called the discount factor [16].

As per Equation (3.1), we can see that the same reward is worth more if received now than if it is received in the future. We can change the behaviour of the agent by changing the discounted rate. When rate is close to 0, it is called “myopic” and it means that the agent is only concerned about immediate reward [16]. If the rate is close to 1, it means that the agent considers future rewards to be more important and future rewards will have more weight.

An agent in RL has to choose an action from the state  $s_k$  provided by the environments. The information given by the environment is called the state signal [16]. The agent needs this information from the present state  $s_{k+1}$ , and the previous states  $s_k$  making the best decision possible and maximizes its rewards. A state signal has Markov properties if it has all the necessary information to define the entire history of the past states.

The agent has all the information needed with the immediate state. The agent does not need to know every past move to choose its next action. In other words, if we can predict the next state and the next expected reward given the current state and the current reward with a probability of  $\rho = \Pr\{s_{t+1} = s'; r_{t+1} = r \mid s_t, a_t\}$  for all  $s', r, s_t$  and  $a$ , and at then the environment has the Markov property[16].

RL algorithms build a model from the data; called the “model learning” [33]. RL algorithms can be further subcategorized, according to the path taken to find an optimal policy. These three subcategories are as follows:

- (i) Value iteration algorithms search for the optimal value function, which consists of the maximal returns from every state or from every state-action pair. The optimal value function is used to compute an optimal policy.
- (ii) Policy iteration algorithms evaluate policies by constructing their value functions (instead of the optimal value function), and these value functions to find new, improved policies.
- (iii) Policy Search algorithms use optimization techniques to directly search for an optimal policy.

Within each of these three subcategories of RL algorithms, two subsequent categories can be further distinguished, namely offline and online. Offline RL algorithm uses data collected in advance, whereas RL algorithm learns a solution by interacting with the process. Online RL algorithm are typically not provided with any future data, but instead depends only on the data collected while learning and hence are useful when data is difficult or costly to obtain in advance. Most online RL algorithm work incrementally.

### 3.4 MARKOV DECISION PROCESS (MDP)

RL problems can be formalized with the help of markov decision process (MDPs) [6]. An RL problem that satisfies the Markov property is called a Markov decision process, or MDP [16]. When the state and action spaces are finite, then it is a finite Markov decision process (finite MDP). Finite MDPs are particularly valuable to the theory of RL. A particular finite MDP is defined by its state and action sets and by the one-step dynamics of the environment. For a particular state and action,  $s$  and  $a$ , the probability of a particular possible next state,  $s_{k+1}$ , is

$$\rho_{ss'}^a = \Pr\{s_{k+1} = s' \mid s_k = s, a_k = a\} \quad (3.2)$$

These variables are called transition probabilities. Similarly, for any present state and action,  $s$  and  $a$ , alongwith any next state,  $s_{k+1}$ , the expected value for next reward is

$$R_{ss'}^a = E\{r_{k+1} \mid s_k = s, a_k = a, s_{k+1} = s'\} \quad (3.3)$$

These variables,  $\rho_{ss'}^a$  and  $R_{ss'}^a$ , completely specify the most vital aspects of the dynamics of a finite MDP.

### 3.5 TEMPORAL DIFFERENCE (TD) LEARNING

TD learning is undoubtedly identified as one idea as central and novel to reinforcement learning [16]. This is a combination of Monte Carlo ideas and dynamic programming ideas. TD methods can learn directly from new experience without a model of the environment's dynamics like Monte Carlo methods. TD approaches update estimates based in portion on other learned estimates, without waiting for a final result. The link between TD, DP, and Monte Carlo methods is a repetitive theme in the theory of RL. The  $TD(\lambda)$  algorithm seamlessly integrates TD and Monte Carlo methods.

For finding the optimal policy in the control problems, DP, TD, and Monte Carlo methods all use some variation of generalized policy iteration (GPI). The differences in these methods are primarily differences in their approaches to the prediction problem.

#### 3.5.1 TD Prediction

Monte Carlo and TD methods use experiences to solve the prediction problems. State some experiences for following a policy  $\pi$ , both update their estimates  $V$  of

$V^\pi$ . If a non-terminal state is visited at time  $k$ , then both the methods update their estimate based on what happens after the visit. Monte Carlo method is suitable for a simple early-visit nonstationary environments is

$$V(s_k) \leftarrow V(s_k) + \alpha [R_k - V(s_k)] \quad (3.4)$$

where,  $R_k$  is the actual return following time  $t$  and  $\alpha$  is a constant step-size parameter. Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to  $V(s_k)$ , TD methods wait only until the next time step. At time  $k+1$  they immediately form a target and make a useful update using the observed reward  $r_{k+1}$  and the estimate  $V(s_{k+1})$ . The simplest TD method, known as TD(0), is

$$V(s_k) \leftarrow V(s_k) + \alpha [r_{k+1} + \gamma V(s_{k+1}) - V(s_k)] \quad (3.5)$$

In effect, Monte Carlo updates the reward as  $R_k$ , whereas the target for the TD update is  $r_{k+1} + \gamma V(s_{k+1})$ .

Because the TD method bases its update in part on an existing estimate, so like DP it is a bootstrapping method.

$$V^\pi(s) = E_\pi\{R_k | s_k = s\} \quad (3.6)$$

$$= E_\pi\{r_{k+1} + \gamma V^\pi(s_{k+1}) | s_k = s\} \quad (3.7)$$

Monte Carlo methods use an estimate of (3.6) as a target, whereas DP methods use an estimate of (3.7) as a target. The Monte Carlo target is an estimate because the expected value in (3.6) is not known; a sample return is used in place of the real expected return. The TD target is an estimate for both reasons: it samples the expected values in (3.7) and it uses current estimates instead of the true  $V^\pi$ . Thus, TD methods combine the sampling of Monte Carlo with the bootstrapping of DP. This can take us a long way toward obtaining the advantages of both Monte Carlo and DP methods.

Algorithm 3.1 specifies TD(0) completely in procedural form. The value estimate for the top node state node of the backup diagram is updated on the basis of the single sample transition from it to the next following state. These updates are referred here as sample backups because they involve looking ahead to a sample successor state, using

the value of the successor and the reward along the way to compute a backed-up value, and later changing the value of the original state accordingly. Sample backups differ from the full backups of DP methods in that they are based on one sample successor rather than on complete distribution of all the possible successors.

Algorithm 3.1: TD(0) method for estimating  $V^\pi$  :-

1. Initialize  $V(s)$  arbitrarily,  $\pi$  to the policy to be evaluated

2. Repeat (for each episode):

Initialize  $s$

Repeat (for each step of episode) :

$a \leftarrow$  action given by  $\pi$  for  $s$

Take action  $a$ ; observe the reward,  $r$ , and next state,  $s'$

$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$

$s \leftarrow s'$

until  $s$  is terminal

### 3.5.2 Optimality of TD (0)

For any problem with finite amount of experiences available 10 episodes or 100 time steps, it is a common approach with the incremental learning method to present the experience recurrently until the method converges. Given an approximate value function,  $V$ , the increments specified by (3.4) or (3.5) are computed for every time step at which a non-terminal state is visited, but the value function is changed only once, by the summation of all increments. Then all the available experiences are processed again with the next value function to produce a new overall increment, until the value function converges. This is called batch updating because updates are made only after processing each of the complete batches of training data.

Under batch updating, TD(0) converges deterministically to one answer independent

of the step size parameter,  $\alpha$ , as long as  $\alpha$  is chosen to be sufficiently small. The constant  $\alpha$ -MC method also converges deterministically under the same conditions, but to a different answer. Under normal updating, the methods do not move all the way to their respective batch answers, but in a manner they take steps in these directions.

Finally, it should be noted that although the certainty-equivalence estimate is in some sense an optimal solution, but it is almost never feasible to compute it directly [16]. If  $N$  is the number of states, then forming the maximum-likelihood estimates of the process may require  $N^2$  memories, and computing the corresponding value function will require an order of  $N^3$  computational steps if done conventionally. It can thus be said that TD methods can approximate the same solution using memory no more than and repeated computations over the training set. For tasks with large state spaces, TD methods might be the only feasible way of approximating the certainty- equivalence solution [16].

### 3.5.3 Value Functions & The Bellman Equations

Policies can be conveniently characterized by their value functions. There are two types of value functions: state-action value functions; Q-functions and state-value functions; V-functions. The Q-function  $Q^\pi: X \times U \rightarrow \mathbb{R}$  of a policy  $\pi$  gives the return obtained when starting from a given state, applying a given action, and following  $\pi$  thereafter:

$$Q^\pi(s,a) = \rho(s,a) + \gamma R^\pi(f(s,a)) \quad (3.8)$$

Here,  $R^\pi(f(s,a))$  is the return from the next state  $f(s,a)$ . This formula can be obtained by first writing  $Q^\pi(s,a)$  explicitly as the discounted sum of rewards obtained by taking  $a$  in  $s$  and then following  $\pi$ :

$$Q^\pi(s,a) = \sum_{k=0}^{\infty} \gamma^k \rho(s_k, a_k) \quad (3.9)$$

where  $(s_0, a_0) = (s, a)$ ,  $s_{k+1} = f(s_k, u_k)$  for  $k \geq 0$  and  $a_k = \pi(s_k)$  for  $k \geq 0$ . The first term is separated from the sum:

$$\begin{aligned} Q^\pi(s,a) &= \rho(s, a) + \sum_{k=1}^{\infty} \gamma^k \rho(s_k, a_k) \\ &= \rho(s,a) + \gamma R^\pi(f(s,a)) \end{aligned} \quad (3.10)$$

The optimal Q-function is defined as the best Q-function that can be obtained by any policy:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad (3.11)$$

Any policy  $\pi^*$  that selects at each state an action with the largest optimal Q-value, i.e., that satisfies:

$$\pi^*(s) \in \arg^* \max_a Q^*(s,a) \quad (3.12)$$

is optimal (it maximizes the return). In general, for a given Q-function  $Q$ , a policy  $\pi$  that satisfies:

$$\pi(s) \in \arg^* \max_a Q(s,a) \quad (3.13)$$

is said to be greedy in  $Q$ . So, finding an optimal policy can be done by first finding  $Q^*$ , and then using (3.12) to compute a greedy policy in  $Q^*$ .

For the computation of greedy actions in (3.12), (3.13), and in similar equations in the sequel, the maximum must exist to ensure the existence of a greedy policy; this can be guaranteed under certain technical assumptions [2].

The Q-functions  $Q^{\pi}$  and  $Q^*$  are recursively characterized by the Bellman equations, which are of core importance for value iteration and policy iteration algorithms. The Bellman equation for  $Q^{\pi}$  states that the value of taking an action;  $a$  in the state;  $s$  under the policy;  $\pi$  is the summation of the immediate reward and the discounted value achieved by  $\pi$  in the next state:

$$Q^{\pi}(s,a) = \rho(s,a) + \gamma Q^{\pi}(f(s,a), \pi(f(s,a))) \quad (3.14)$$

The Bellman optimality equation characterizes  $Q^*$ , and states that the optimal value of action  $a$  taken in state,  $s$  is the summation of the immediate reward and the discounted optimal value obtained by the best action in the next state:

$$Q^*(s,a) = \rho(s,a) + \gamma \max_{a'} Q^*(f(s,a), a') \quad (3.15)$$

The V-function  $V^{\pi} : S \rightarrow \mathbb{R}$  of a policy  $\pi$  is the return obtained by starting from a particular

state and following  $\pi$ . This V-function can be computed from the Q-function of policy  $\pi$ :

$$V^\pi(s) = R^\pi(s) = Q^\pi(s, \pi(s)) \quad (3.16)$$

The optimal V-function is the best V-function that can be obtained by any policy, and can be computed from the optimal Q-function:

$$V^*(s) = \max_\pi V^\pi(s) = \max_a Q^*(s, a) \quad (3.17)$$

An optimal policy  $\pi^*$  can be computed from  $V^*$ , by using the fact that it satisfies:

$$\pi^*(s) \in \arg \max_a [\rho(s, a) + \gamma V^*(f(s, a))] \quad (3.18)$$

Using this formula is more difficult than using (3.12); in particular, a model of the MDP is required in the form of the dynamics  $f$  and the reward function  $\rho$ . Because the Q-function also depends on the action, it already includes information about the quality of transitions. In contrast, the V-function only describes the quality of the states; in order to infer the quality of transitions, they must be explicitly taken into account. This is what happens in (3.18), and this also explains why it is more difficult to compute policies from V-functions.

### 3.5.4 Q-Learning: Off-Policy TD Control

#### 3.5.4.1. Model-based Q-iteration algorithm

Model-based Q-iteration algorithm is an illustrative example from the class of model-based value iteration algorithms. Let the set of all the Q-functions be denoted by  $\mathcal{N}$ . Then, the Q-iteration mapping  $T : \mathcal{N} \rightarrow \mathcal{N}$ , computes the right-hand side of the Bellman optimality equation (3.15) for any Q-function. In the deterministic case, this mapping is:

$$[T(Q)](s, a) = \rho(s, a) + \gamma \max_{a'} Q^*(f(s, a), a') \quad (3.15)$$

and in the stochastic case, it is:

$$[T(Q)](s, a) = E_{s' \sim f(s, a, \cdot)} \{ \rho(s, a, s') + \gamma \max_{a'} Q^*(s', a') \} \quad (3.16)$$

If the state space is countable (e.g., finite) then the Q-iteration mapping for the stochastic case (3.15) can be written as the simpler summation:



$$[T(Q)](s,a) = \sum_{s'} f(s,a,s') [\rho(s,a,s') + \gamma \max_{a'} Q^*(s',a')] \quad (3.17)$$

The same notation is used for the Q-iteration mapping both in the deterministic case and in the stochastic case, because the analysis given below applies to both cases, and the definition (3.15) of  $T$  is a special case of (3.16).

The Q-iteration algorithm starts from an arbitrary Q-function  $Q_0$  and for each iteration  $k$  updates the Q-function using:

$$Q_{k+1} = T(Q_k) \quad (3.18)$$

It can be shown that  $T$  is a contraction with factor  $\gamma < 1$  in the infinity norm, i.e., for any pair of functions  $Q$  and  $Q'$ , it is true that:

$$\|T(Q) - T(Q')\|_{\infty} \leq \gamma \|Q - Q'\|_{\infty} \quad (3.19)$$

Because  $T$  is a contraction, it has a unique fixed point. Additionally, when rewritten using the Q-iteration mapping, the Bellman optimality equation (3.15) states that  $Q^*$  is a fixed point of  $T$ , i.e.:

$$Q^* = T(Q^*) \quad (3.20)$$

Hence, the unique fixed point of  $T$  is actually  $Q^*$ , and Q-iteration asymptotically converges to  $Q^*$  as  $k \rightarrow \infty$ . Moreover, Q-iteration converges to  $Q^*$  at a rate of  $\gamma$ , in the sense that  $\|Q_{k+1} - Q^*\|_{\infty} \leq \gamma \|Q_k - Q^*\|_{\infty}$ . An optimal policy can be computed from  $Q^*$  with (3.12).

Algorithm 3.2 presents Q-iteration for deterministic MDPs in an explicit, procedural form, wherein  $T$  is computed using (3.15). Similarly, algorithm 3.3 presents Q-iteration for stochastic MDPs with countable state spaces, using the expression (3.17).

Algorithm 3.2: Q-Iteration for deterministic MDPs :-

Input: dynamics  $f$ , reward function  $\rho$ , discount factor  $\gamma$

1. Initialize Q function as  $Q_0 \leftarrow 0$
2. Repeat at every iteration  $k = 0, 1, 2, \dots$

3. for every (s,a) do

$$Q_{k+1}(s,a) \leftarrow \rho(s,a) + \gamma \max_{a'} Q^*(f(s,a), a')$$

end for

until  $Q_{k+1} = Q_k$

Output:  $Q^* = Q_k$

Algorithm 3.3: Q-Iteration for stochastic MDPs with countable state spaces :-

Input: dynamics  $f$ , reward function  $\rho$ , discount factor  $\gamma$

1. Initialize Q function as  $Q_0 \leftarrow 0$

2. Repeat at every iteration  $k = 0, 1, 2, \dots$

3. for every (s,a) do

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} f(s,a,s') [\rho(s,a,s') + \gamma \max_{a'} Q^*(s', a')]$$

end for

until  $Q_{k+1} = Q_k$

Output:  $Q^* = Q_k$

### 3.5.4.2. Model-free value iteration and the need for exploration

Q-learning starts from an arbitrary initial Q-function  $Q_0$  and updates it without requiring a model, using instead observed state transitions and rewards, i.e., data tuples of the form  $(s_k, a_k, s_{k+1}, r_{k+1})$  [4]. After each transition, the Q-function is updated using such a data tuple, as follows:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k [r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k)] \quad (3.21)$$

where,  $\alpha_k \in (0,1]$  is the learning rate. The term between square brackets is the temporal difference, i.e., the difference between the updated estimate  $r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a')$  of the optimal Q-value of  $(s_k, a_k)$ , and the current estimate  $Q_k(s_k, a_k)$ . In the deterministic case, the

new estimate is actually the Q-iteration mapping (3.15) applied to  $Q_k$  in the state-action pair  $(s_k, a_k)$ , where  $r(s_k, a_k)$  has been replaced by the observed reward  $r_{k+1}$ , and  $f(s_k, a_k)$  by the observed next-state  $s_{k+1}$ .

In the stochastic case, these replacements provide a single sample of the random quantity whose expectation is computed by the Q-iteration mapping (3.16), and thus Q-learning can be seen as a sample-based, stochastic approximation procedure based on this mapping [12].

In practice, the learning rate schedule may require tuning, because it influences the number of transitions required by Q-learning to obtain a good solution. A good choice for the learning rate schedule depends on the problem at hand. The controller also has to exploit its current knowledge in order to obtain good performance, e.g., by selecting greedy actions in the current Q-function. This is a typical illustration of the exploration-exploitation trade-off in online RL. A classical way to balance exploration with exploitation in Q-learning is  $\epsilon$ -greedy exploration [16], which selects actions according to:

$$a_k = a \in \arg \max_a Q_k(s_k, a), \quad \text{with probability } 1 - \epsilon_k$$

$$\text{a uniformly random action in } A, \quad \text{with probability } \epsilon_k \quad (3.22)$$

where,  $\epsilon_k \in (0,1)$  is the exploration probability at step  $k$ . Another option is to use Boltzmann exploration[16], which at step  $k$  selects an action  $a$  with probability:

$$P(a|s_k) = (e^{Q(s_k, a)/\tau_k}) / \sum_a (e^{Q(s_k, a)/\tau_k}) \quad (3.23)$$

where the temperature  $\tau_k \geq 0$  controls the randomness of the exploration. When  $\tau_k \rightarrow 0$ , (3.23) is equivalent to greedy action selection, while for  $\tau_k \rightarrow \infty$ , action selection is uniformly random. For nonzero, finite values of  $\tau_k$ , higher-valued actions have a greater chance of being selected than lower-valued ones.

Usually, the exploration diminishes over time, so that the policy used asymptotically becomes greedy and therefore optimal. This can be achieved by making  $\tau_k$  or  $\epsilon_k$  approach 0 as  $k$  grows. For instance, an  $\epsilon$ -greedy exploration schedule of the form  $\epsilon_k = 1/k$  diminishes to 0 for  $k \rightarrow \infty$ , while still satisfying the convergence condition of Q-

learning, i.e., while allowing infinitely many visits to all the state-action pairs [18]. Like the learning rate schedule, the exploration schedule has a significant effect on the performance of Q-learning. Algorithm 3.4 presents Q-learning with  $\epsilon$ -greedy exploration.

Algorithm 3.4: Q-Iteration with  $\epsilon$ -greedy exploration :-

Input: discount factor  $\gamma$ , exploration schedule  $\epsilon_k$ , learning rate schedule  $\alpha_k$

1. Initialize Q function as  $Q_0 \leftarrow 0$

2. Measure initial state

3. for every time step  $k = 0, 1, 2, \dots$  do

$$a_k \leftarrow \begin{cases} a \in \arg \max_a Q_k(s_k, a) & \text{with probability } 1 - \epsilon_k \text{ (exploit)} \\ \text{a uniformly random action in } A & \text{with probability } \epsilon_k \text{ (explore)} \end{cases}$$

apply  $a_k$ , measure next state  $s_{k+1}$  and reward  $r_{k+1}$

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k [r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k)]$$

end for

### 3.5.5 SARSA: On-Policy TD Control

Another class of RL, model-free policy iteration algorithms is SARSA, an online algorithm proposed by Rummery and Niranjan [8] as an alternative to the value-iteration based Q-learning. The name SARSA is obtained by joining together the initials of every element in the data tuples employed by the algorithm, namely: state, action, reward, (next) state, (next) action. Formally, such a tuple is denoted by  $(s_k, a_k, r_{k+1}, s_{k+1}, a_{k+1})$ . SARSA starts with an arbitrary initial Q-function  $Q_0$  and updates it at each step using tuples of this form, as follows:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k [r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1}) - Q_k(s_k, a_k)] \quad (3.24)$$

where  $\alpha_k \in (0, 1]$  is the learning rate. The term between square brackets is the temporal difference, obtained as the difference between the updated estimate  $r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1})$  of

the Q-value for  $(s_k, a_k)$ , and the current estimate  $Q_k(s_k, a_k)$ . This is not the same as the temporal difference used in Q-learning (3.21). While the Q-learning temporal difference includes the maximal Q-value in the next state, the SARSA temporal difference includes the Q-value of the action actually taken in this next state. This means that SARSA performs online, model-free policy evaluation of the policy that is currently being followed. In the deterministic case, the new estimate  $r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1})$  of the Q-value for  $(s_k, a_k)$  is actually the policy evaluation mapping (3.15) applied to  $Q_k$  in the state-action pair  $(s_k, a_k)$ . Here,  $r(s_k, a_k)$  has been replaced by the observed reward  $r_{k+1}$ , and  $f(s_k, a_k)$  by the observed next state  $s_{k+1}$ . In the stochastic case, these replacements provide a single sample of the random quantity whose expectation is found by the policy evaluation mapping (3.16). Next, the policy employed by SARSA is considered.

As in offline policy iteration, SARSA cannot afford to wait until the Q-function has (almost) converged before it improves the policy. This is so because convergence may take a long time, during which the unchanged (and possibly bad) policy would be implemented. Instead of this, to select actions, SARSA combines a greedy policy in the current Q-function with exploration, using, e.g.,  $\epsilon$ -greedy (3.22) or Boltzman (3.23) exploration. Because of the greedy component, SARSA implicitly performs a policy improvement at every iterative step, and is thus a type of online policy iteration. Such a policy iteration algorithm, which improves the policy after every sample, is sometimes called fully optimistic [12].

Algorithm 3.5 presents SARSA with  $\epsilon$ -greedy exploration. In this algorithm, because the update at step  $k$  involves the action  $a_{k+1}$ , this action has to be chosen prior to updating the Q-function.

In order to converge to the optimal Q-function  $Q^*$ , SARSA requires conditions similar to those of Q-learning, which demand exploration, and additionally that the exploratory policy being followed asymptotically becomes greedy [18]. Such a policy can be obtained by using, e.g.,  $\epsilon$ -greedy (3.22) exploration with an exploration probability  $\epsilon_k$  that asymptotically decreases to 0, or Boltzmann (3.23) exploration with an exploration temperature  $\tau_k$  that asymptotically decreases to 0. The exploratory policy used by Q-

learning can also be made greedy asymptotically, even though the convergence of Q-learning does not rely on this condition.

Algorithm 3.5: SARSA with  $\epsilon$ -greedy exploration :-

Input: discount factor  $\gamma$ , exploration schedule  $\epsilon_k$ , learning rate schedule  $\alpha_k$

1. Initialize Q function as  $Q_0 \leftarrow 0$

2. Measure initial state  $s_0$

3.  $a_0 \leftarrow \begin{cases} a \in \arg \max_a Q_0(s_0, a) & \text{with probability } 1 - \epsilon_0 \text{ (exploit)} \\ \text{a uniformly random action in } A & \text{with probability } \epsilon_0 \text{ (explore)} \end{cases}$

4. for every time step  $k = 0, 1, 2, \dots$  do

    apply  $a_k$ , measure next state  $s_{k+1}$  and reward  $r_{k+1}$

$a_k \leftarrow \begin{cases} a \in \arg \max_a Q_k(s_k, a) & \text{with probability } 1 - \epsilon_k \text{ (exploit)} \\ \text{a uniformly random action in } A & \text{with probability } \epsilon_k \text{ (explore)} \end{cases}$

    apply  $a_k$ , measure next state  $s_{k+1}$  and reward  $r_{k+1}$

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k [r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1}) - Q_k(s_k, a_k)]$$

end for

Algorithms like SARSA, which evaluate the policy are also called “on-policy” in the RL literature [16]. In contrast, algorithms like Q-learning, which act on the process using one policy and evaluate another policy, are called “off-policy.”

### 3.6 LIMITATIONS OF RL

Though its numerous advantages and applications, RL methods do have certain limitations and disadvantages. Some of the limitations that RL methods have, are stated as follows:

- As no model for is provided for any given problem, the model that is estimated by the RL methods has shallow knowledge and might sometimes make the agent to take a wrong action.
- In RL methods, the reward calculated for the overall goal is taken and the final output is compared then rather than comparing the output at each step or caomparing the best possible output at each step. This restricts the agent form looking ahead.
- Reinforcement Learning has two parameters for learning called the learning rate and the exploration rate. If the learning rate is not properly chosen, it might take the agent a longer time to come to an optimal solution and thus can restrict its ability to learn And if the exploration rate is not set properly, it might happen that it takes too long for agent to reach the goal or it may never reach the goal also.

## CHAPTER 4

### MULTI-AGENT GRID WORLD PROBLEM

#### 4.1 GRID WORLD PROBLEM

The most widely used benchmark problem in reinforcement learning is Grid world [16]. In its most basic form, this domain consists of a discrete planar grid with finite dimensions. An agent is placed at some set starting location, and then selects cardinal actions (up, down, left, right) to move within the grid with the goal of reaching some specific goal grid location. Modifications to this domain include addition of diagonal moves, the addition of penalty or hole grid locations, changes to the dimension of the grid, and addition of a stochastic wind component that acts on the agent [16]. Further extensions include using a large grid space with multiple rooms. Yet another modification could be dynamic obstacles, in which the obstacle emerges for random locations in the grid and also the goal could be made moving or dynamic by specifying an area where the goal emerges randomly or moves step wise in a particular group of grid cells.

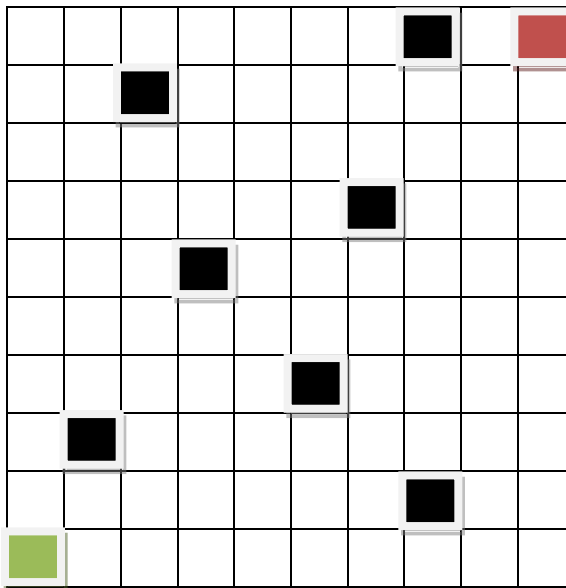


Figure 4.1 A sample grid world with green block as obstacles, red block as goal and green block as starting block

All these modifications are purely environmental and have an effect on the of the



domain, which has downstream effects on the actions of the agent, the efficacy of the learning algorithm, and finally on the knowledge acquired by the agent and its performance in the domain.

The grid world problem used in this thesis is an evolving one. Initially, a single agent case of simple 10 X 10 grid is taken and very few fixed obstacles are present. The number of fixed obstacles are added starting from 4 obstacles to 15 obstacles. Later on, few of the obstacles are made to emerge randomly in the grid. Another complexity that has been studied is the grid world problem is the case of moving goal. The upper two rows of the grid are made as the area in which the goal can emerge randomly. This randomness has been done for each iteration movement of the agent and not for the epochs.

#### 4.1.1 Single Agent Problem

In single-agent RL, the environment of the agent is described by a MDPs. “A finite Markov decision process is a tuple  $\{S, A, f, \rho\}$  where  $S$  is the finite set of environment states,  $A$  is the finite set of agent actions,  $f : S \times A \times S \rightarrow [0, 1]$  is the state transition probability function, and  $\rho : S \times A \times S \rightarrow \mathbb{R}$  is the reward function”[49]. The state signal  $s_k \in S$  describes the environment at each discrete time-step  $k$ . The agent can alter the state at each time step by taking actions  $a_k \in A$ . As a result of the action  $a_k$ , the environment changes its state from  $s_k$  to some  $s_{k+1} \in S$  according to the state transition probabilities given by  $f$ : the probability of ending up in  $s_{k+1}$  given that  $a_k$  is executed in  $s_k$  is  $f(s_k, a_k, s_{k+1})$ . The agent receives a scalar reward  $r_{k+1} \in \mathbb{R}$ , according to the reward function  $\rho$ :  $r_{k+1} = \rho(s_k, a_k, s_{k+1})$  [16]. This reward evaluates the immediate effect of action  $a_k$ , i.e., the transition from  $s_k$  to  $s_{k+1}$ . It however, says nothing directly about the long-term effects of this action.

For deterministic models, the transition probability function  $f$  is replaced by a simpler transition function,  $f : S \times A \rightarrow S$ . It follows that the reward is completely determined by the current state and action:  $r_{k+1} = \rho(s_k, a_k)$ ,  $\rho : S \times A \rightarrow \mathbb{R}$ .

The behavior of the agent is described by its policy  $\pi$ , which specifies how the agent chooses its actions given the state. The policy may be either stochastic,  $\pi : S \times A \rightarrow [0, 1]$ , or deterministic,  $\pi : S \rightarrow A$ . A policy is called stationary if it does not change over time [31]. The task of the agent is, to maximize its long-term performance, while only

receiving feedback about its immediate, one-step performance. One way it can achieve this is by computing an optimal action-value function.

#### 4.1.2 Multi - Agent Problem

The generalization of the Markov decision process to the multi agent case is the stochastic game. “A stochastic game (SG) is a tuple  $(S, A_1, \dots, A_n, f, \rho_1, \dots, \rho_n)$ , where  $n$  is the number of agents,  $S$  is the discrete set of environment states,  $U_i, i = 1, \dots, n$  are the discrete sets of actions available to the agents, yielding the joint action set  $A = A_1 \times \dots \times A_n$ ,  $f: S \times A \times S \rightarrow [0, 1]$  is the state transition probability function, and  $\rho_i: S \times A \times S \rightarrow \mathbb{R}, i = 1, \dots, n$  are the reward functions of the agents.

In the multi agent case, the state transitions are the result of the joint action of all the agents,  $\mathbf{a}_k = [a_{1,k}^T, \dots, a_{n,k}^T]^T, a_k \in A, a_{i,k} \in A_i$  (T denotes vector transpose). Consequently, the rewards  $r_{i,k+1}$  and the returns  $R_{i,k}$  also depend on the joint action. The policies  $\pi_i: S \times A_i \rightarrow [0, 1]$  form together the joint policy  $\pi$ . The  $Q$ -function of each agent depends on the joint action and is conditioned on the joint policy,  $Q_i^\pi: S \times A \rightarrow \mathbb{R}$  [31]. If  $\rho_1 = \dots = \rho_n$ , all the agents have the same goal (to maximize the same expected return), and the SG is fully cooperative. If  $n = 2$  and  $\rho_1 = -\rho_2$ , the two agents have opposite goals, and the SG is fully competitive. Full competition can also arise when more than two agents are involved. In this case, the reward functions must satisfy  $\rho_1(s, \mathbf{a}, s') + \dots + \rho_n(s, \mathbf{a}, s') = 0 \forall s, s' \in S, \mathbf{a} \in A$ . However, the literature on RL in fully competitive games typically deals with the two-agent case only. Mixed games are stochastic games that are neither fully cooperative nor fully competitive.

#### 4.2 Multi – Agent Reinforcement Learning (MARL)

Various MARL algorithms are: Minimax-Q Learning, Nash-Q Learning, Friend-or-Foe Q Learning and Win-or-Learn-Fast Policy Hill Climbing (WOLF-PHC)[31]. These algorithms represent an evolution from the Q-Learning algorithm and provide an insight into multi-agent learning. Minimax-Q learning algorithm is one of the first adaptations of the original Q-learning algorithm and is still in use. For Nash-Q Learning algorithm Nash Equilibrium is the basis for convergence. It was designed to reach a Nash Equilibrium strategy between two fully competitive players. The Friend-or-Foe Q Learning

algorithm was designed from the Nash-Q Learning and updated for an environment where the learning has friends and/or foes.

The Minimax-Q learning algorithm is interesting because it takes the single agent Q-learning and adapts it for a multi-agent environment. It also uses the linear programming to maximize its rewards by minimizing its opponents rewards. In a zero sum stochastic game, the Minimax-Q learning algorithm will converge and find the Nash equilibrium strategy. It was not proven to converge in general-sum games and it is a limitation. A general-sum game environment can give more flexibility because the rewards do not need to respect  $R1 = -R2$  where  $R1$  and  $R2$  are the rewards for Player 1 and Player 2.

The different assumptions that make Nash-Q learning a restrictive algorithm. It was proven to converge within these assumptions, but it cannot be generalized for every general-sum game. In [12], Littman discussed the limitations of the Nash-Q Learning algorithm. In the Nash-Q, updates taken into account is either the global optimal point or the saddle point, but there are occasions where both are present. It was shown in [3] and [30] that the algorithm does converge even if not all the assumptions are respected.

Friend or Foe-Q (FFQ) has been designed to alleviate the flaws of Nash-Q Learning when confronted with the coordination and adversarial equilibrium. Littman [12] discussed this by pointing out that there is a possibility that both equilibriums exist at the same time. This can create problems in Nash-Q because it is not designed to decide which one to choose. In FFQ there is a selection mechanism and it can alleviate this problem.

#### **4.2.1 Friend Or Foe Algorithm**

This algorithm was developed by Littman and it tries to fix some of the convergence problems of Nash-Q Learning. The convergence of Nash-Q is restricted by various assumptions made during the solving of a problem. The main concern lies within assumptions, where every stage game needs to have either a global optimal point or a saddle point. These restrictions cannot be guaranteed during learning. To ease this restriction, the FFQ algorithm is built and it always converges by changing the update rules subjected to the agents. The learning agent has to identify the other agents either as “friend” or “foe”.

The FFQ algorithm is built for the n-player game, but we will start with a two player game to understand this concept. One of the main differences between the Nash-Q and the FFQ is that the agent only keeps track of its own Q-table. The update performed by the agent is given by the following equation:

$$\max_{a_i \in A_i} Q_i[s, a_1, \dots, a_n] \quad (4.1)$$

when the opponents are friends; and

$$\max_{\pi \in \Pi(A_i)} \min_{a_i \in A_i} \sum_{a_i \in A_i} Q_i[s, a_1, \dots, a_n] \quad (4.2)$$

when the opponents are foes, where n is the number of agents and i is the learning agent [18].

Equation (4.1) is the Q-Learning algorithm adapted for multiple agents and Equation (4.2) is the minimax-Q algorithm from Littman [6]. These equations represent a situation where all the agents are either friend or foe.

We can categorize as all the agents in this algorithm in two groups of people i's friends and i's foes. The friends will work together to maximize i's payoff. The foes will work together against i to minimize its pay-off. The algorithm 3.1 showing friend or foe Q-learning is as follows:

#### 1. Initialization

$\forall s \in S, a_1 \in A_1$  and  $a_2 \in A_2$

Let  $Q(s, a_1, a_2) = 0 \forall s \in S$

Let  $V(s) = 0 \forall s \in S, a_1 \in A_1$

Let  $\pi(s, a_1) = 1/|A_1|$

for  $k = 0, 1, 2, \dots$  do

In state  $s$ : Choose a random action from  $A_1$  with probability  $\pi$

If not a random action, choose action  $a_1$  with probability  $\pi(s, a_1)$

Evaluate the next state,  $s'$ .

In state  $s'$  : The agent observes the reward  $r$  related to action  $a_1$  and opponent's action  $a_2$  in state  $s$ . Update Q-Table of player 1 with equation :

$$Q(s, a_1, a_2) \leftarrow Q(s, a_1, a_2) + \alpha [(r + \gamma V(s')) - Q(s, a_1, a_2)]$$

Use linear programming to find the values of  $Q(s; a_1)$  and  $V(s)$  with the equation

$$V(s) = \max_{a_i \in A_i} Q_i[s, a_1, \dots, a_n] \quad \text{if the opponent is a friend, and;}$$

$$V(s) = \max_{\pi \in \Pi(A_i)} \min_{a_i \in A_i} \sum_{a_i \in A_i} Q_i[s, a_1, \dots, a_n] \quad \text{if the opponent is a foe}$$

$$\alpha = \alpha * \text{decay}$$

End loop

## CHAPTER 5

### HYBRID REINFORCEMENT LEARNING

#### 5.1 INTRODUCTION

In ordinary RL algorithms with a single agent, the agent often takes a useless action with a small reward, which results in a long learning time. On the other hand, in swarm optimization algorithms, multiple agents are prepared and some agents could take useful actions with a larger reward. Similarly, in with ant colony optimization, agents can be made to learn which actions result in overall higher rewards. In addition, since the Q values of all the agents are updated according to Q-values of such agents who take the useful actions, it is expected that agents can learn in a shorter learning time.

#### 5.2 RL & PARTICLE SWARM OPTIMIZATION (PSO)

RL has been recently combined with Particle Swarm Optimization to overcome the problems while evaluating the multi-agent reinforcement learning for a dynamic environment [16, 17]. The navigation problem for a single agent system with static system provides quick control for the conventional RL methods such as Q learning and SARSA. When subjected to a multi-agent system, the algorithm works well till the system dynamics are less. This is so because with epochs, the RL algorithm focusses less on exploring new paths and try to converge to the predefined goal for which the path was searched previously. When the goal is changing the robots do not communicate that efficiently with each other to change their termination points. With the PSO algorithm, as there are two parameters on which each agent takes any action, they are able to communicate in a better way. The personal best (or personal minima in PSO) of an individual agent drives it to reach to their personal best path and simultaneously the global best of all the agents (or global minima in PSO) also drives the agent to search and take the global best path.

##### 5.2.1 Q - Swarm

Q-Swarm is a combination of Q-learning method and Particle Swarm Optimization Algorithms. First, each agent updates its own Q-values individually by using Q-learning for some episodes. Then, the Q-values of all the agents are evaluated by an

adequate method, and the Q-values evaluated superior to those of the other agents are selected. We call them the best Q values. Then, each agent receives the best Q-values from another agent, and updates its own Q-values according to an adequate strategy by using the best Q-values. These procedures are repeated until a termination condition is satisfied.

Although the Q-values are not evaluated in ordinary Q-learning, this algorithm requires to evaluate them in order to select the best Q-values which bring a large reward. In the Q-Swarm algorithm, there are two kinds of procedures of updating the Q-values of each agent. One is the procedure of Q-learning, which is performed in the inner loop. The other is the procedure based on interaction among the agents, which is performed in the outer loop by the following equation. To evaluate the best of all Q-values discounted reward is used [16]:

$$E = \sum_{k=1}^N r^k d^{N-k} \quad (5.1)$$

The personal best of each agent  $i$ ;  $P_i$  and the global best found by all the agents till time;  $G$  are determined by evaluating  $E$  for the Q-values. Each agent updates its Q-values by using the global best and its personal best using the following equations [16]:

$$V_i(s,a) \leftarrow W V_i(s,a) + C_1 R_1 (P_i(s,a) - Q_i(s,a)) + C_2 R_2 (G(s,a) - Q_i(s,a)) \quad (5.2)$$

$$Q_i(s,a) \leftarrow Q_i(s,a) + V_i(s,a) \quad (5.3)$$

where,  $V_i(s,a)$  is a so-called velocity,  $W$ ,  $C_1$  and  $C_2$  are weight parameters, and  $R_1$  and  $R_2$  are uniform random numbers in the range 0 to 1. In this algorithm, there are two kinds of procedural for updating the Q-values of each agent. One is the procedure of Q-learning, which is performed in the inner loop. The other is the procedure based on the interaction among the agents, which is performed in the outer loop. The algorithm 5.1 shows the Q-swarm as follows.

Algorithm 5.1: Q - Swarm learning[16]

Input: discount factor  $\gamma$ , Number of episode  $Y$ , number of agents  $n$

1. For the agents  $i = 1, 2, \dots, n$  initialize  $Q_i$  function as  $Q_{i0} \leftarrow 0$

2. Measure initial state,  $s_0$

3. for all the agents, For every time step  $k = 1, 2, \dots$  do

$$a_k \leftarrow \begin{cases} a \in \arg \max_a Q_k(s_k, a) & \text{with probability } 1 - \epsilon_k \text{ (exploit)} \\ \text{a uniformly random action in } A & \text{with probability } \epsilon_k \text{ (explore)} \end{cases}$$

apply  $a_k$ , measure next state  $s_{k+1}$  and reward  $r_{k+1}$

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_n [r_{k+1} + \gamma \max_{a'} (Q_k(s_{k+1}, a')) - Q_k(s_k, a_k)]$$

If a terminate condition of episode is satisfied, go to step 4.

end for

4. Calculate the evaluated  $E$  for  $Q_i(s, a)$  of each agent by equation 5.1

5. Update  $Q_i(s, a)$  of each agent by applying an information exchange using equation 5.2 and 5.3.

6. Go to step 3 till global termination criteria is not reached.

### 5.2.2 SARSA - Swarm

SARSA-Swarm is a combination of SARSA method and Particle Swarm Optimization algorithm [30]. This algorithm is similar to the Q-swarm algorithm and has been proposed by the same authors. This algorithm is best suited for environment with large negative rewards. An environment in which the reward for travelling around is negative for any action not reaching the goal instead for an environment in which the same reward is zero then in the first case, the steps needed to reach the goal is faster. And for such an environment, the SARSA reinforcement learning algorithm gives the best result [30].

In such an environment, if a group of agents or robots are to be navigated, then the simple SARSA algorithm does not prove to be fruitful. Thus, SARSA algorithm coupled with the swarm optimization gives a swarm movement and the agents



communicate and cooperate with each other to reach the goal.

Algorithm 5.2: SARSA - Swarm learning [30] :

Input: discount factor  $\gamma$ , Number of episode  $Y$ , number of agents  $n$

1. For the agents  $i = 1, 2, \dots, n$  initialize  $Q_i$  function as  $Q_{i0} \leftarrow 0$

2. Measure initial state,  $s_0$

3.  $a_0 \leftarrow \begin{cases} a \in \arg \max_a Q_0(s_0, a) & \text{with probability } 1 - \epsilon_0 \text{ (exploit)} \\ \text{a uniformly random action in } A & \text{with probability } \epsilon_0 \text{ (explore)} \end{cases}$

4. for all the agents, For every time step  $k = 0, 1, 2, \dots$  do

$a_{k+1} \leftarrow \begin{cases} a \in \arg \max_a Q_{k+1}(s_{k+1}, a) & \text{with probability } 1 - \epsilon_{k+1} \text{ (exploit)} \\ \text{a uniformly random action in } A & \text{with probability } \epsilon_{k+1} \text{ (explore)} \end{cases}$

apply  $a_k$ , measure next state  $s_{k+1}$  and reward  $r_{k+1}$

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_n [r_{k+1} + \gamma Q_k(s_{k+1}, a_{k+1}) - Q_k(s_k, a_k)]$$

If a terminate condition of episode is satisfied, go to step 4.

end for

4. Calculate the evaluated  $E$  for  $Q_i(s, a)$  of each agent by equation 5.1

5. Update  $Q_i(s, a)$  of each agent by applying an information exchange using equation 5.2 and 5.3.

6. Go to step 3 till global termination criteria is not reached.

### 5.3 RL & Ant Colony Optimization (ACO)

Ants are able to find the shortest route between the nest and a food source [3] without any vision [10]. This process is possible because ants secrete pheromone chemicals

on the trail as they cover the path while hunting for food or resources to construct a nest. Initial exploration is random in the absence of a pheromone trail. Ants returning to the nest secrete pheromone on the trail. The pheromone evaporates with time; but ants follow a pheromone trail and at a splitting point prefer to navigate the path with higher concentrations of pheromone. Upon finding the food source, the ants return back to the nest depositing pheromone along the way, thus reinforcing the pheromone trail.

Ants that have followed the shortest route are quicker to return to the nest, thus reinforcing the pheromone concentration for the shorter trail at a quicker rate than those ants that followed an alternative route. Further, when ants arrive at the branching point, it chooses to follow the path which has the higher concentrations of pheromone, and thus reinforces even further the pheromone concentration, and ultimately all ants follow the shortest path. The quantity of pheromone secreted is a function of an angle between the path and a line joining the food and nest locations [1] on the return journey. So far two properties of pheromone secretion have been mentioned: aggregation and evaporation. The concentration adds when ants deposit pheromone at the same location, and over time the concentration gradually reduces by evaporation. A third property is diffusion. The pheromone at one location diffuses into neighbouring locations.

Some of the mechanisms adopted by foraging ants have been applied to classical combinatorial optimization problems with success. These problems include the travelling salesman problem, job-shop scheduling, the quadratic assignment problem, the vehicle routing problem and the network routing problem, robot navigation problem [19, 23, 32].

### **5.3.1 Pheromone-Q Learning**

The Pheromone-Q technique is a combination of Q-learning and synthetic pheromone where a belief factor is introduced in the update [8]. The belief factor is a function of the synthetic pheromone concentration on the trail and shows the extent to which an agent takes into account the information laid down by other agents from the same cooperating set. RL and synthetic pheromone have previously been combined for action selection [14, 15].

The belief factor allows an agent to selectively make use of implicit communication trails which have been left by other agents; this might be useful in situations where the information is not reliable due to changes in the environment. Incomplete and uncertain informations are critical issue in the design of real-world systems.

### 5.3.1.1 Belief Factor

The belief factor directs the extent to which an agent believes in the pheromone it detects. Any agent, during the early training episodes, will believe less in the pheromone map because then all the agents are biased towards exploration. In practical terms, the belief factor is the ratio between the sum of actual pheromone concentration in the current state plus neighbouring states and the sum of maximum possible pheromone concentration in the current plus neighbouring states [10]. As such the value for the belief factor falls in the range [0,1]. The belief factor is given by

$$B(s,a) = \sum_{s \in N_a} \Phi(s) / \sum_{\sigma \in N_a} \Phi_{\max}(\sigma) \quad (5.1)$$

where,  $\Phi(s)$  is the pheromone concentration at a point  $s$  in the environment and  $N_a$  is the set of neighbouring states for a chosen action  $a$ . The belief factor is a function of the synthetic pheromone  $\Phi(s)$ , a scalar value that integrates the basic dynamic nature of the pheromone, namely aggregation, evaporation and diffusion.

The Q-learning update equation modified with synthetic pheromone is given by

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_n [r_{k+1} + \Upsilon [\max_{a'} Q_k(s_{k+1}, a') + \xi B(s_{k+1}, a')] - Q_k(s_k, a_k)] \quad (5.2)$$

and, 
$$\alpha_n = 1/(1 + \text{visits}_n(s_k, a_k)) \quad (5.2a)$$

where, the parameter  $\xi$  is a sigmoid function of time epochs  $\geq 0$  and  $\text{visits}(s,a)$  is the total number of times the state-action pair is visited. The value of the parameter  $\xi$  increases with the number of agents who successfully perform the task at hand. The Phe-Q update equation converges for a non-deterministic Markov decision process [10].

The parameter  $\Upsilon$  is the discount factor and reflects the relative strength of delayed reward to immediate reward. The value for  $\alpha$  is given by equation (5.2a). The

corresponding parameters in the Phe-Q update equation use the optimum values found for the standard Q-learning algorithm. The parameters that influence Phe-Q learning are the number of agents, the diffusion rate, secretion rate, evaporation rate and the coefficients of the sigmoid of the pheromone and finally the pheromone saturation level [8]. The pheromone distribution in the environment is a function of the number of existing agents, and also a function of the diffusion across cells and the evaporation.

The agents moves from cell to cell along the four directions and secretes synthetic pheromone in each cell. The two pheromone values – one associated with the return to the nest  $\phi_n$  and the other with search for the food source  $\phi_s$  – are parameters to fine tune. The pheromone aggregates in a cell up to a saturation level, and evaporates at an evaporation rate  $\phi_e$ ; until there is no pheromone left in the cell. Also, the pheromone diffuses into neighbouring cells at a rate with diffusion rate  $\phi_d$  which is inversely proportional to the distance.

Each agent has two tasks. First is to reach the food location, and other is to return to the nest. When released into the virtual environment, the agents have no knowledge of the environment or the location of the goal. More than one agent can occupy a cell. A cell has associated pheromone strength  $\Phi \in [0, 100]$ . Pheromone is decoupled from the state at the implementation level so that the size of the state space is  $N \times N$ . Algorithm 5.3 which shows the Phe-Q is as follows:

Algorithm 5.3: Phe-Q learning

Input: discount factor  $\gamma$ ,

1. Initialize Q function as  $Q_0 \leftarrow 0, B \leftarrow 0$
2. Measure initial state,  $s_0$
3. for every time step  $k = 1, 2, \dots$  do

$$a_k \leftarrow \begin{cases} a \in \arg \max_a Q_k(s_k, a) & \text{with probability } 1 - \epsilon_k \text{ (exploit)} \\ a \text{ uniformly random action in } A & \text{with probability } \epsilon_k \text{ (explore)} \end{cases}$$

apply  $a_k$ , measure next state  $s_{k+1}$  and reward  $r_{k+1}$

Update the pheromone value  $\Phi(s_{k-1})$  and the pheromone table for the previous state:

$$\Phi(s_{k-1}) = (\varphi_s + \varphi_n + \varphi_d) e^{\varphi_s}$$

$$B(s,a) = \sum_{s \in \mathcal{N}_a} \Phi(s) / \sum_{\sigma \in \mathcal{N}_a} \Phi_{\max}(\sigma)$$

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_n [r_{k+1} + \gamma \max_{a'} (Q_k(s_{k+1}, a') + \xi B(s_{k+1}, a')) - Q_k(s_k, a_k)]$$

$$\alpha_n = 1 / (\text{visits}_n(s_k, a))$$

end for

### 5.3.2 Pheromone-SARSA Learning

RL methods have been used with many evolutionary algorithms according to the need of the applications. It has been seen in the Phe-Q algorithm that by combining the belief factor in the Q-updation rule changes the way agents work and hence with cooperative actions, the agent reach the goals faster in search problems[26]. Also, it is evident that for a negative reward environment, SARSA method provides a better result than the Q-learning method.

In literature so far, these two algorithms has not been combined to solve the various search problems. In this thesis, a new method called the Pheromone-SARSA or Phe-SARSA is introduced. This is similar to the Phe-Q algorithm only with the basic Q-updation rule is that followed in the SARSA.

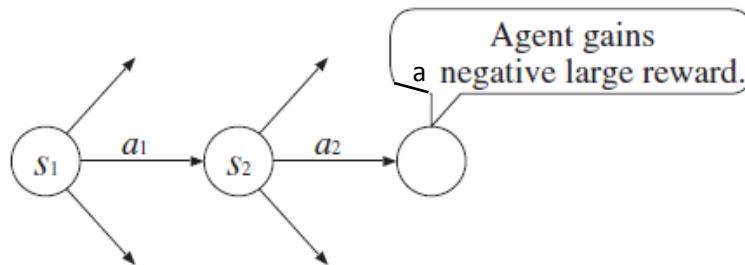


Figure 5.1 Environment in which SARSA is effective

Q-learning is the most frequently used reinforcement method but it is basically and offline learning schedule. The advantage of SARSA over Q-learning can be appreciated

when working with large negative rewards.

The main differences between SARSA method and Q-learning method can be explained using a simple example. Let an agent in state  $s_1$  perceive the next state  $s_2$  by taking action  $a_1$ , and gain a negative large reward by taking the next action  $a_2$  in  $s_2$ , as shown in Fig.5.1. In this figure, a circle and an arrow mean a state and an action, respectively. For both, SARSA method and Q-learning method, as  $Q(s_2, a_2)$  becomes a large negative value by the large negative large reward, the agent learns that  $a_2$  is a bad selection. Moreover, in SARSA method, because  $Q(s_1, a_1)$  also tends to become a large negative value in future episodes, the agent can learn that  $a_1$  is not a good selection. Therefore, it can avoid such actions and acquire a better policy rapidly. Instead in Q-learning method, because other Q-values in  $s_2$  are generally larger than  $Q(s_2, a_2)$ ,  $Q(s_1, a_1)$  is updated without using  $Q(s_2, a_2)$  in the future episodes. Hence, the agent cannot learn that  $a_1$  is a bad selection.

Here, the Phe-SARSA method is proposed in order to obtain an optimal policy rapidly for problems with negative large rewards. In this algorithm, the basic framework is the same as the SARSA with a modified updating rule for the Q-value in which a belief factor is also updated. Algorithm 5.4 shows the proposed Phe-SARSA algorithm:

Algorithm 5.4: Phe-SARSA learning

Input: discount factor  $\gamma$ , exploration schedule  $\epsilon_k$

1. Initialize Q function as  $Q_0 \leftarrow 0$ ,  $B \leftarrow 0$
2. Measure initial state,  $s_0$
3.  $a_0 \leftarrow \begin{cases} a \in \arg \max_a Q_0(s_0, a) & \text{with probability } 1 - \epsilon_0 \text{ (exploit)} \\ a \text{ uniformly random action in } A & \text{with probability } \epsilon_0 \text{ (explore)} \end{cases}$
3. for every time step  $k = 1, 2, \dots$  do
4. apply  $a_k$ , measure next state  $s_{k+1}$  and reward  $r_{k+1}$

$$5. \quad a_k \leftarrow \begin{cases} a \in \arg \max_a Q_k(s_k, a) & \text{with probability } 1 - \epsilon_k \text{ (exploit)} \\ \text{a uniformly random action in } A & \text{with probability } \epsilon_k \text{ (explore)} \end{cases}$$

apply  $a_k$ , measure next state  $s_{k+1}$  and reward  $r_{k+1}$

Update the pheromone value  $\Phi(s_{k-1})$  and the pheromone table for the previous state:

$$\Phi(s_{k-1}) = (\varphi_s + \varphi_n + \varphi_d) e^{\varphi_s}$$

$$B(s, a) = \sum_{s' \in N_a} \Phi(s') / \sum_{\sigma \in N_a} \Phi_{\max}(\sigma)$$

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_n [r_{k+1} + \gamma \max_{a'} (Q_k(s_{k+1}, a') + \xi B(s_{k+1}, a')) - Q_k(s_k, a_k)]$$

$$\alpha_n = 1 / (\text{visits}_n(s_k, a))$$

end for

## CHAPTER 6

### SIMULATION RESULT AND DISCUSSION

A 10 X 10 grid has been taken and initially one agent was trained for various conditions. The number of agent was gradually increased from one to two and finally to four. The different environments for which the simulation has been carried out can be broadly divided into four cases and in each case further variations have been introduced. Different cases for which the various algorithms have been implemented are as follows:

- Case I: Obstacles: Fixed; Goal: Fixed
  - Case I(a): No. of Obstacles : 4
  - Case I(b): No. of Obstacles : 8
  - Case I(c): No. of Obstacles : 14
- Case II: Obstacles: Fixed and Moving (Both); Goal: Fixed
  - Case II(a): No. of Fixed Obstacles : 4; No. of Moving Obstacles: 2
  - Case II(b): No. of Fixed Obstacles : 5; No. of Moving Obstacles: 3
- Case III: Obstacles: Fixed; Goal: Moving
  - Case III(a): No. of Fixed Obstacles : 4
  - Case III(b): No. of Fixed Obstacles : 8
  - Case III(c): No. of Fixed Obstacles : 14
- Case IV: Obstacles: Fixed and Moving (Both); Goal: Moving
  - Case IV(a): No. of Fixed Obstacles : 4; No. of Moving Obstacles: 2
  - Case IV(b): No. of Fixed Obstacles : 5; No. of Moving Obstacles: 3

Agents can move in four directions: up, down, left and right. In all the cases, the starting location of the agent(s) was the bottom leftmost corner grid and the goal location when fixed was the top rightmost corner grid. The moving obstacles could move in four directions: up, down, left and right, and for the goal, it could move in a band of first two rows as shown by a patch. Termination criterion used for the simulations were:-

- Maximum number of steps in an attempt is 5000.
- Maximum number of attempts is 2500.







Reward function that has been taken is :

$$R(x_k, x_{k+1}) = \begin{cases} +100, & \text{when } x_k \neq \text{goal and } x_{k+1} = \text{goal} \\ -100, & \text{when } x_{k+1} \text{ is an obstacle or wall} \\ -1, & \text{otherwise} \end{cases}$$

where,  $x_k$  is the current state and  $x_{k+1}$  is the next state.

For each of the various cases stated above, 100 simulations were carried out and the sum average of these simulations is shown in the graphs between numbers of steps needed to reach the goal vs. number of attempts.

In the images shown below for paths travelled by the agents have been shown, following legends was followed:

-  --Starting location of Agents
-  -- Goal/ End Point for Agents
-  -- Fixed Obstacles
-  -- Moving Obstacles

Programming Software used – MATLAB 2013a

System Configuration: Pentium (R) Dual – Core @2.10 GHz, 2.00 GB RAM, 32-bit Operating System

## 6.1 Single Agent Problem

For the single agent case, following results was obtained for the various environmental cases:

### 6.1.1 Case I: Obstacles Fixed, Goal Fixed

(a) No. of obstacles = 4

Figure 6.1.1 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 4 obstacles in the path. For the single agent problem, two conventional RL methods, Q-learning and SARSA and two hybrid-RL methods, i.e. RL with ACO, Phe-Q and Phe-SARSA have been simulated.

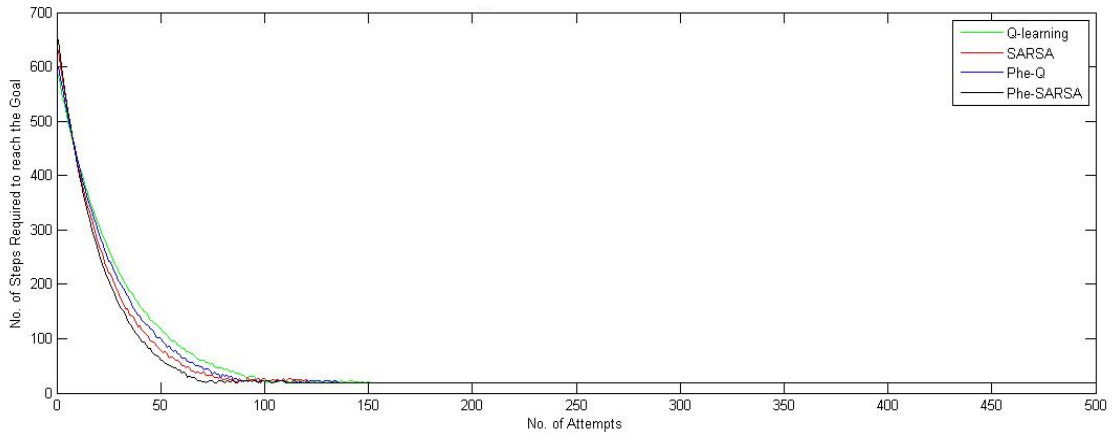


Figure 6.1.1 Plot between no. of steps required to reach the goal and no. of attempts for 1 agent; case I(a)

Figure 6.1.2a, 6.1.2b, 6.1.2c, 6.1.2d shows the grid world with the path traced by the agent using Q-learning, SARSA, Phe-Q and Phe-SARSA, respectively for four obstacles in path.

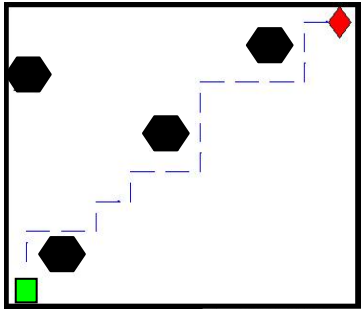


Figure 6.1.2a Path traced for case I(a) by single agent for Q-learning

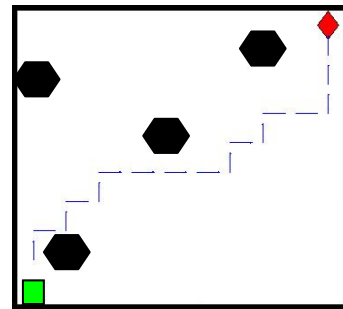


Figure 6.1.2b Path traced for case I(a) by single agent for SARSA

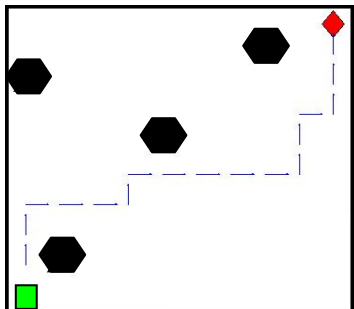


Figure 6.1.2c Path traced for case I(a) by single agent for Phe-Q

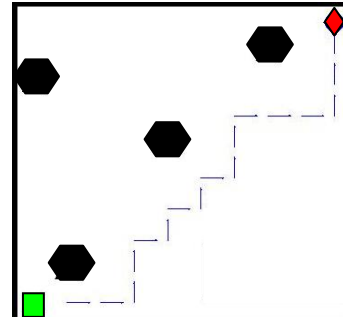


Figure 6.1.2d Path traced for case I(a) by single agent for Phe-SARSA

Here, the agent is very much free to move around and the obstacles are fewer as compared to the cells in which it can move, and hence the path taken by the agent for all the cases are not same to reach the goal. For all the four algorithms; shortest path has 18 numbers of steps, but Phe-SARSA reaches the optimal path fast as compared to the other three methods.

(b) No. of obstacles = 8

Figure 6.1.3 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 8 obstacles in the path. Two conventional RL methods, Q-learning and SARSA and two hybrid-RL methods, i.e. RL with ACO, Phe-Q and Phe-SARSA have been simulated.

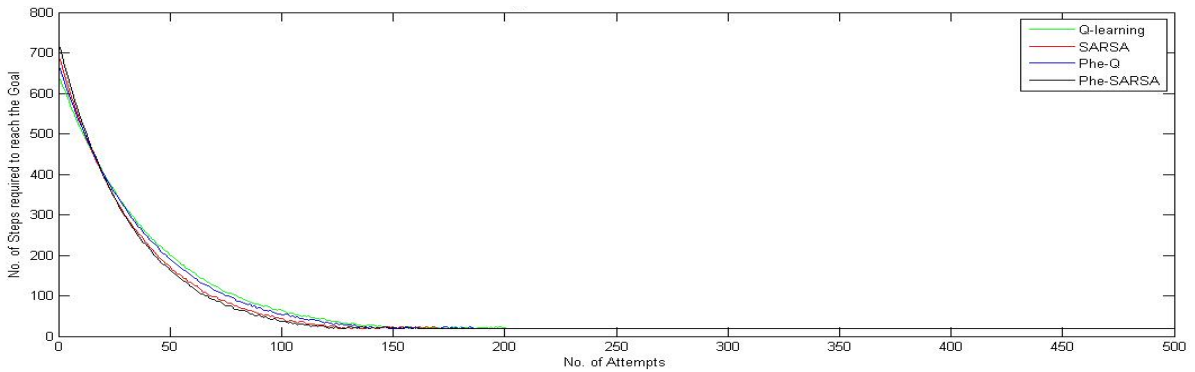


Figure 6.1.3 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case I(b)

Figure 6.1.4a, 6.1.4b, 6.1.4c, 6.1.4d shows the grid world with the path traced by the agent using Q-learning, SARSA, Phe-Q and Phe-SARSA, respectively for eight obstacles in path.

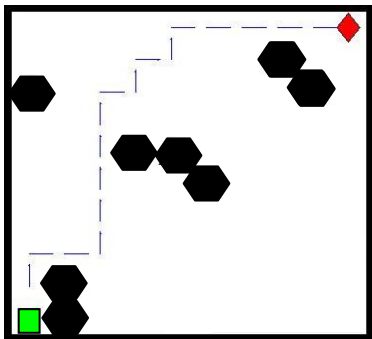


Figure 6.1.4a Path traced for case I(b) by single agent for Q-learning

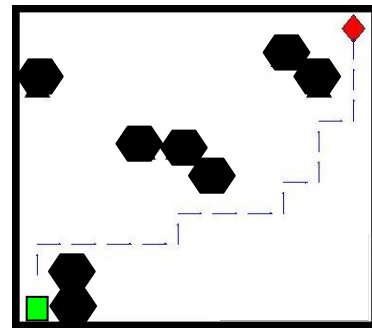


Figure 6.1.4b Path traced for case I(b) by single agent for SARSA

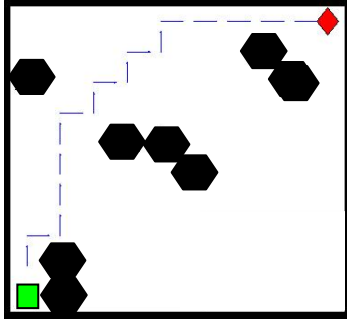


Figure 6.1.4c Path traced for case I(b) by single agent for Phe-Q

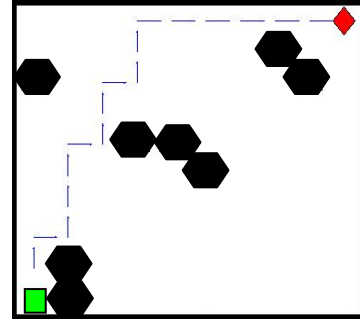


Figure 6.1.4d Path traced for case I(b) by single agent for SARSA

Here also the agent is quite to move around and the obstacles are fewer as compared to the cells to which it can move, and hence the path taken by the agent for all the cases are more or less same to reach the goal. Thus, for all the four algorithms; the shortest path traced has the minimum number of steps as 18 but with Phe-SARSA with the least no. of attempts, this could be found.

(c) No. of obstacles = 14

Figure 6.1.5 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 14 obstacles in the path. Two conventional RL methods, Q-learning and SARSA and two hybrid-RL methods, i.e. RL with ACO, Phe-Q and Phe-SARSA have been simulated.

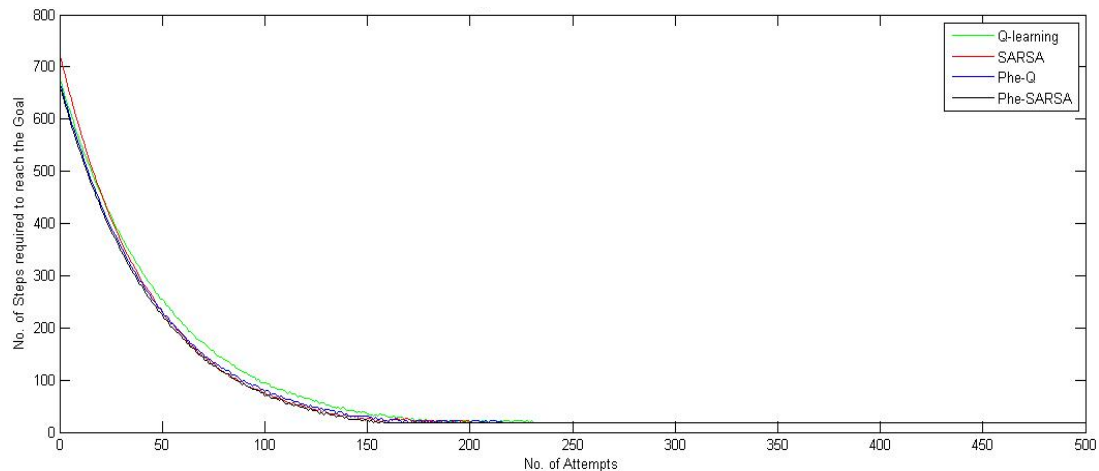


Figure 6.1.5 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case I(c)

Figure 6.1.6a, 6.1.6b, 6.1.6c, 6.1.6d shows the grid world with the path traced by the agent using Q-learning, SARSA, Phe-Q and Phe-SARSA, respectively for fourteen obstacles placed in the path.

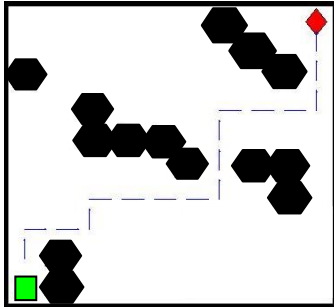


Figure 6.1.6a Path traced for case I(c) by single agent for Q-learning

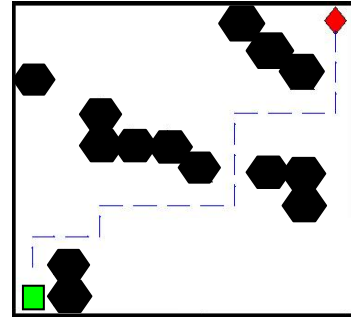


Figure 6.1.6b Path traced for case I(c) by single agent for SARSA

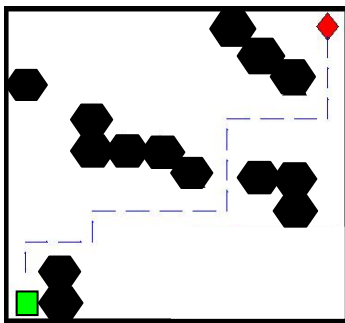


Figure 6.1.6c Path traced for case I(c) by single agent for Phe-Q

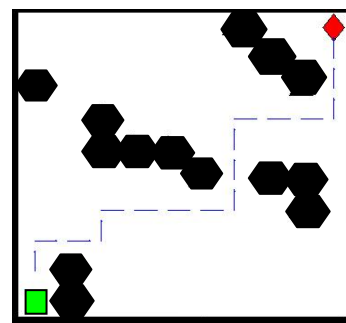


Figure 6.1.6d Path traced for case I(c) by single agent for Phe-SARSA

Here, the agent does not have much area to move around and the obstacles are more as compared to the previous cases, the path taken by the agent for all the cases are same to reach the goal. To reach the goal in the minimum of 18 steps there are only two possible paths. For all the four methods used, agents track down the same shortest path route but with the Phe-SARSA method, the agent could reach the goal with least no. of attempts.

### 6.1.2 Case II: Obstacles Fixed & Moving (Both), Goal Fixed

(a) No. of fixed obstacles = 4; No. of moving obstacles = 2

Figure 6.1.7 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10 X10 grid world with 4 fixed and 2 moving obstacles in the path. Here also

four algorithms have been simulated: Q-learning, SARSA, Phe-Q and Phe-SARSA.

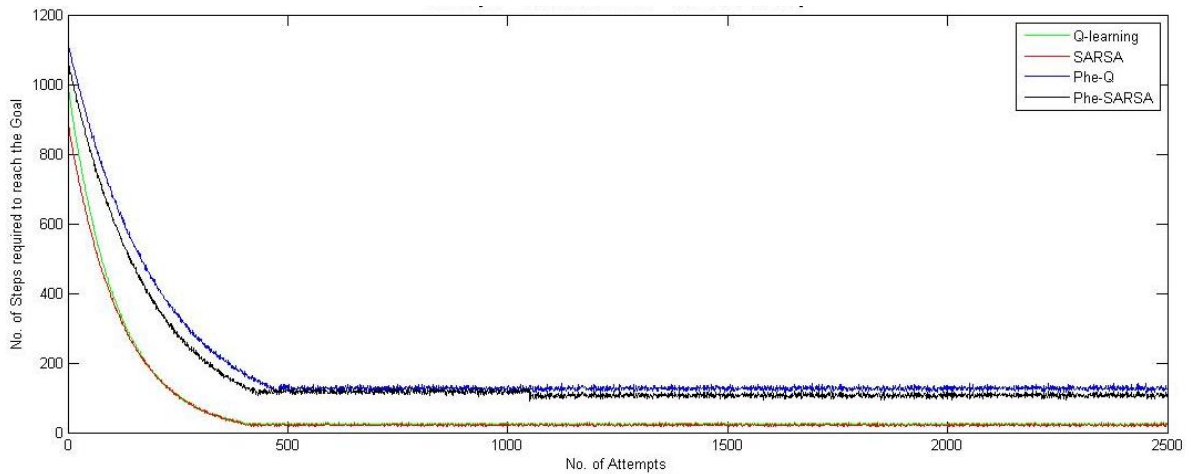


Figure 6.1.7 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case II(a)

Figure 6.1.8a, 6.1.8b shows the grid world with the path traced by the agent using Q-learning and SARSA, respectively for 4 fixed and 2 moving obstacles placed in the path. The fixed obstacles are shown by black hexagon and the moving ones by blue. The cells in which the moving obstacles can move are shown using the grey patch.

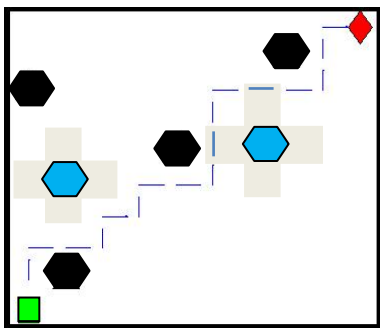


Figure 6.1.8a Path traced for case II(a) by single agent for Q-learning

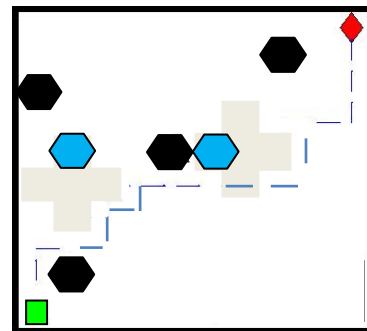


Figure 6.1.8b Path traced for case II(a) by single agent for SARSA

The result shows that for such a case the Q-learning and SARSA converges faster to find their optimal path faster as compared to the hybrid ones. The Phe-Q and Phe-SARSA methods converge slowly and also the final optimal value found by these methods are very large as compared to conventional RL methods. This happens because, with obstacles being dynamic in nature, the pheromone levels create a confusion to decide over which cell to

travel next. This confusion makes the agent to travel in the wrong direction and the overall steps thus taken to reach the goal are very high.

(b) No. of fixed obstacles = 5; No. of moving obstacles = 3

Figure 6.1.9 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 5 fixed and 3 moving obstacles in the path. Here also four algorithms have been simulated: Q-learning, SARSA, Phe-Q and Phe-SARSA.

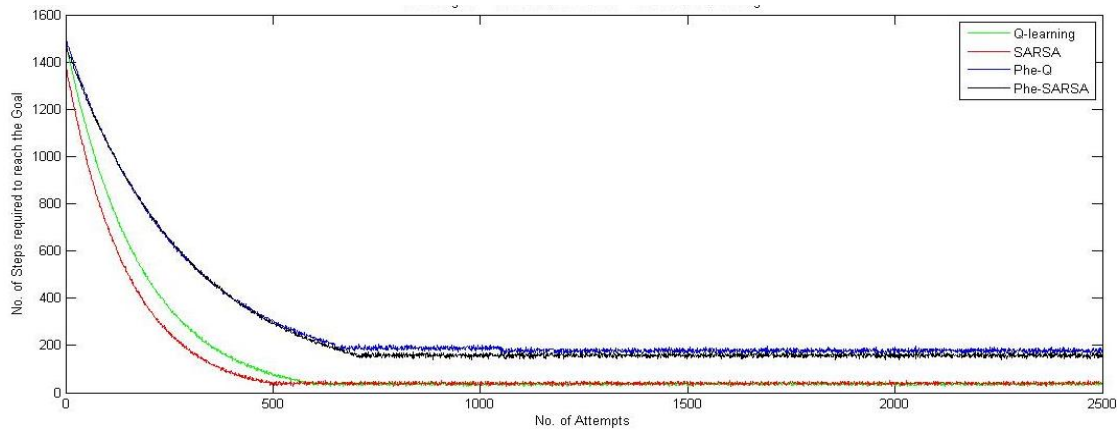


Figure 6.1.9 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case II(b)

Figure 6.1.10a, 6.1.10b shows the grid world with the path traced by the agent using Q-learning and SARSA, respectively for 5 fixed and 3 moving obstacles placed in the path. The fixed obstacles are shown by black hexagon and the moving ones by blue. The cells in which the moving obstacles can move are shown using the grey patch.

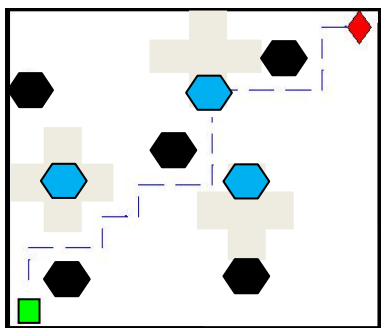


Figure 6.1.10a Path traced for case II(b) by single agent for Q-learning

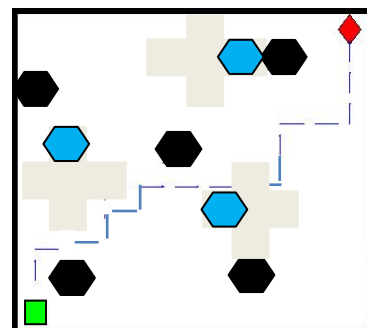


Figure 6.1.10b Path traced for case II(b) by single agent for SARSA

The results obtained here are very similar to that for the last case; Q-learning and SARSA converges faster to find the optimal path as compared to the hybrid ones. The Phe-Q and Phe-SARSA methods converge slowly and the final optimal value found by these methods are very large as compared to conventional RL methods. With the increase in number of obstacles, this gap between conventional and hybrid RL algorithm further increases.

### 6.1.3 Case III: Obstacles Fixed, Goal Moving

(a) No. of fixed obstacles = 4

Figure 6.1.11 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 4 obstacles in the path and the goal is moving randomly for a given set of locations. Four algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA.

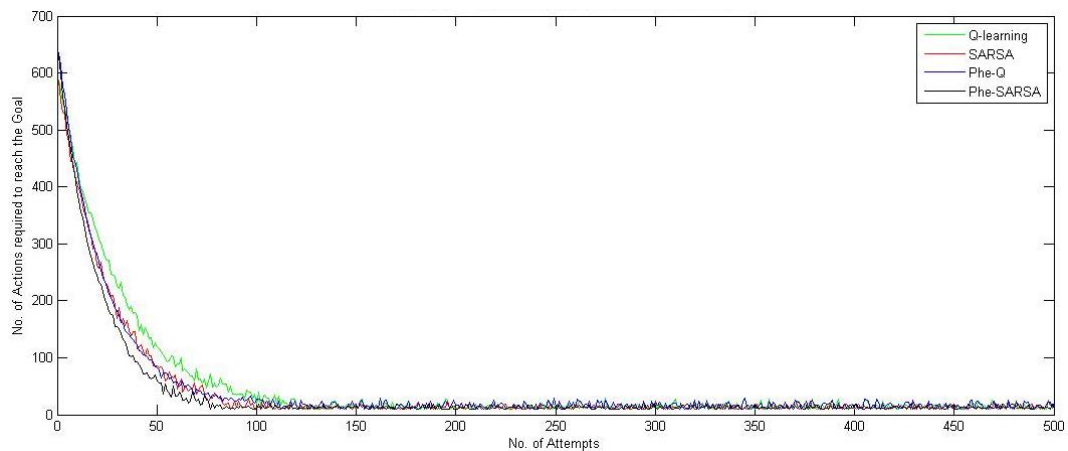


Figure 6.1.11 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case III(a)

Various simulations were done for the four algorithms stated for different goal locations. Figure 6.1.12a, 6.1.12b, 6.1.12c, 6.1.12d shows the grid world with the path traced by the agent (when the goal is moving) for different goal locations and there are 4 obstacles in the path for Q-learning, SARSA, Phe-Q and Phe-SARSA, respectively.

Simulation has been carried out for an environment, in which the goal is moving, the grey cells shows the region in which the goal might appear or is moving for various attempts to reach the goal.



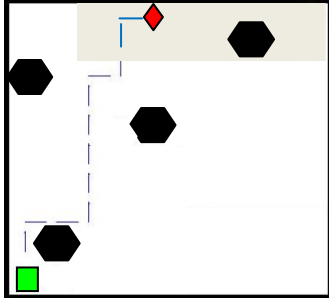


Figure 6.1.12a Path traced for case III(a) by single agent for Q-learning

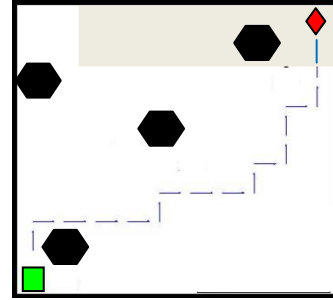


Figure 6.1.12b Path traced for case III(a) by single agent for SARSA

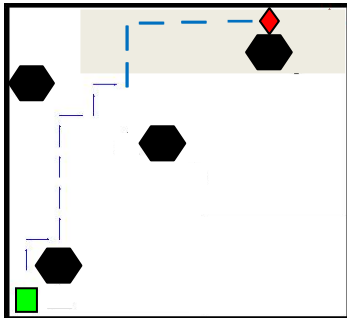


Figure 6.1.12c Path traced for case III(a) by single agent for Phe-Q

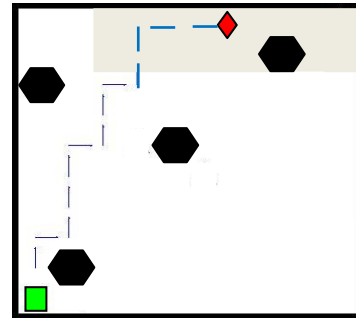


Figure 6.1.12d Path traced for case III(a) by single agent for SARSA

Here, for all the four methods the agent is able to find to optimal paths after various number of attempts. But, Phe-SARSA method is able to find the shortest path for given goal locations in the least number of attempts. This is so because, with Phe-SARSA or Phe-Q, the pheromone levels for the various cells travelled are increased with the number of attempts. As the number of attempts increases, the agents have an understanding of moving vertically to reach the bands where goal can be present and then traverse to look for the actual position of the goal.

(b) No. of fixed obstacles = 8

Figure 6.1.13 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 8 obstacles in the path and the goal is moving in the first two rows. Four algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA.

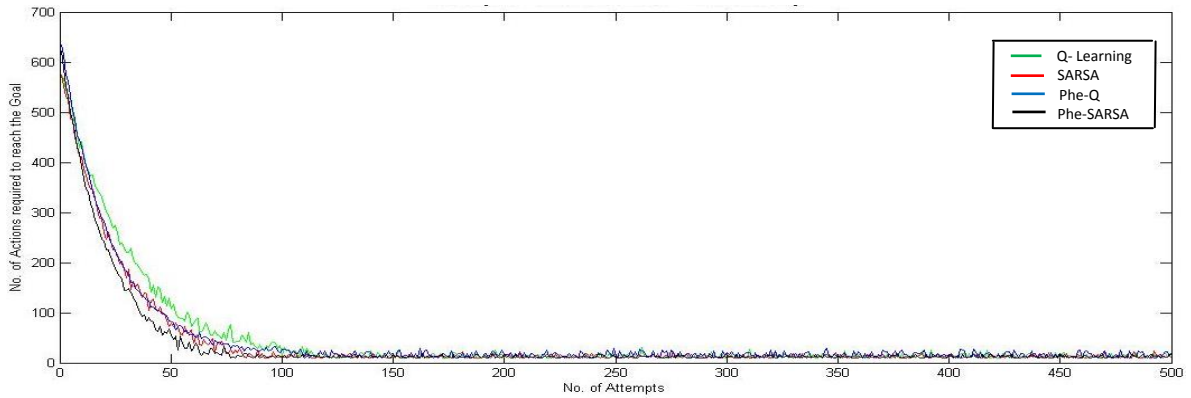


Figure 6.1.13 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case III(b)

Figure 6.1.14a, 6.1.14b, 6.1.14c, 6.1.14d shows the grid world with the path traced by the agent when the goal is moving and there are 8 obstacles in the path for Q-learning, SARSA, Phe-Q and Phe-SARSA.

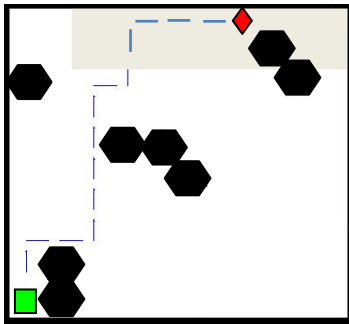


Figure 6.1.14a Path traced for case III(b) by single agent for Q-learning

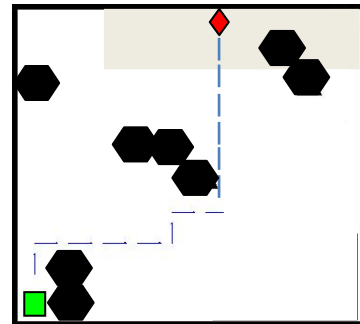


Figure 6.1.14b Path traced for case III(b) by single agent for SARSA

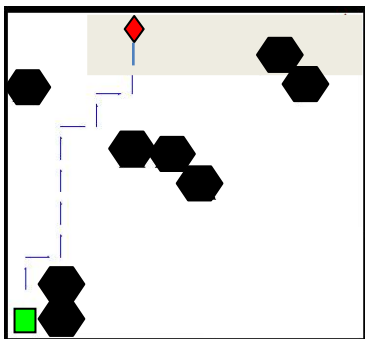


Figure 6.1.14c Path traced for case III(b) by single agent for Phe-Q

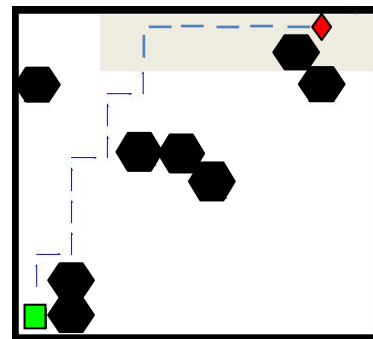


Figure 6.1.14d Path traced for case III(b) by single agent for SARSA

Here also, for all the four methods the agent is able to find to optimal paths after various number of attempts. But, Phe-SARSA method is able to find the shortest path for given goal loactions in the least number of attempts. This is so because, with Phe-SARSA or Phe-Q, the pheromone levels for the various cells travelled are increased with the number of attempts. As the number of attempts increases, the agents have an understanding of moving vertically to reach the bands where goal can be present and then traverse to look for the actual position of the goal.

(c) No. of fixed obstacles = 14

Figure 6.1.15 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 15 obstacles in the path and the goal is moving in the first two rows. Four algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA.

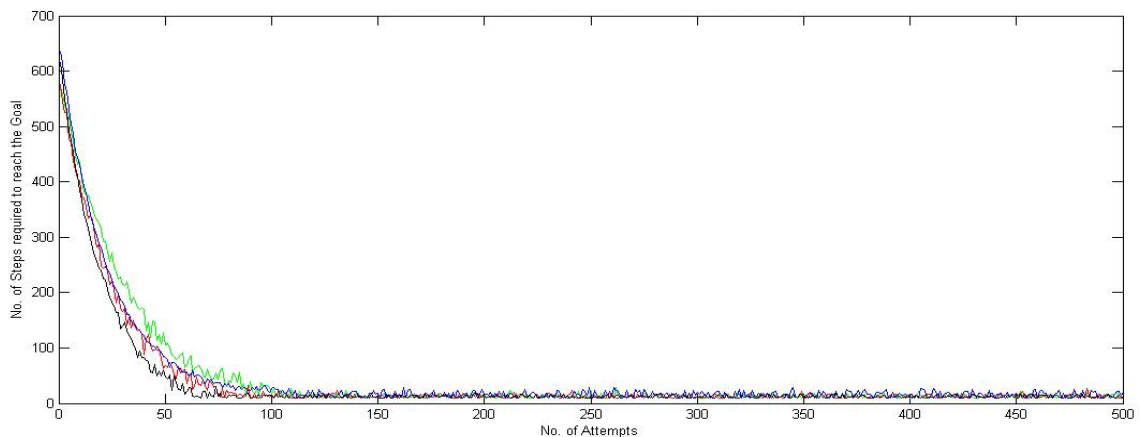


Figure 6.1.15 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case III(c)

Figure 6.1.16a, 6.1.16b, 6.1.16c, 6.1.16d shows the grid world with the path traced by the agent when the goal is moving and there are 14 obstacles in the path for Q-learning, SARSA, Phe-Q and Phe-SARSA, respectively.

It can be seen from the results obtained that for all the four methods the agent is able to find to optimal paths after various number of attempts. But, phe-sarsa method is able to find the shortest path for given goal loactions in the least number of attempts. This is so because, with Phe-SARSA or Phe-Q, the pheromone levels for the various cells travelled are



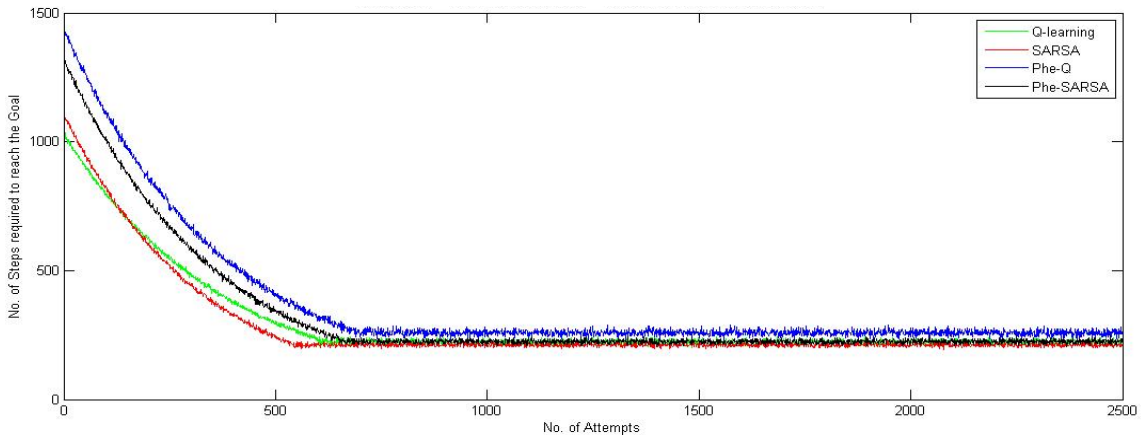


Figure 6.1.17 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case IV(a)

and SARSA, this is so because with goals and obstacles both moving, the pheromone level deposited are almost evenly distributed in the plane and the agent is not able to judge which path to take. Pheromone of almost same levels in the cells creates confusion for the agent and hence even after many attempts, the agent is not able to reach the optimal path as obtained by the Q-learning or SARSA.

(b) No. of fixed obstacles = 5; No. of moving obstacles = 3

Figure 6.1.18 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 5 fixed and 3 moving obstacles in the path and the

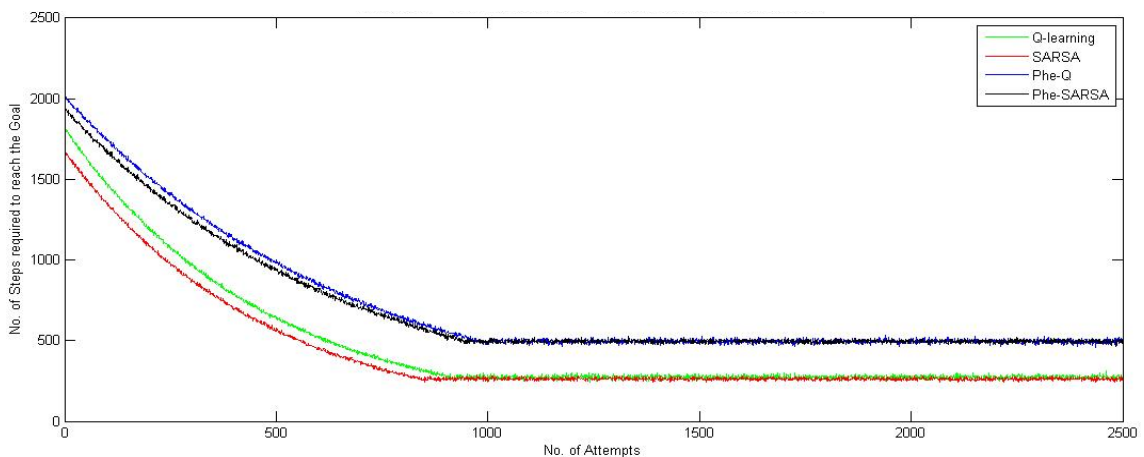


Figure 6.1.18 Plot between No. of Steps required to reach the Goal and No. of Attempts for 1 agent; case IV(b)

goal is moving in the first two rows. Four algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA.

Upon increasing both the number of fixed and moving obstacles, the convergence for the Phe-Q and Phe-SARSA becomes poorer as compared to the other two conventional RL methods. This is so because with the increase in system dynamics there is also less freedom for the agents to move and explore around. As the agent is not able to explore the space properly, it traces the same wrong path again and again and thus ending up for quite high number of steps to reach the goal.

A comparative analysis for the computational time for each of the cases for all four algorithms is provided in following table:

Table 6.1 Computational Time for the Single Agent Problem for the four algorithms simulated

Cases	SARSA	Q-Learning	Phe-Q	Phe-SARSA
I(a)	0.002650692	0.004889478	0.0122237	0.010695733
I(b)	0.002578897	0.004884682	0.0127002	0.012573171
I(c)	0.002627867	0.005205458	0.0145753	0.014385803
II(a)	0.013484825	0.02273135	0.050009	0.005055962
II(b)	0.022228309	0.036542522	0.073085	0.073086754
III(a)	0.005513254	0.008956071	0.0179121	0.017911963
III(b)	0.005707159	0.008127772	0.0227578	0.022025872
III(c)	0.005423821	0.008984571	0.0258756	0.025708512
IV(a)	0.014594556	0.023480353	0.0469607	0.042365314
IV(b)	0.023137526	0.0361615	0.0831715	0.08309836

The computation table shows that the SARSA takes the least time of computation out of all the four algorithms used. Q-learning is slower than SARSA because for Q-learning for the Q updation formula, we use a maximum function which uses more memory for computation whereas in SARSA such a function is not required. The overall time computation for the Phe-Q or Phe-SARSA is around two times that of the conventional RL, this is so because for these methods the ant needs to reach the goal and come back again to the initial starting location before the next simulation is carried out.

## 6.2 Two Agents Problem

For the two agents problem, following results was obtained for the various environmental cases:

### 6.2.1 Case I: Obstacles Fixed, Goal Fixed

(a) No. of obstacles = 4

Figure 6.2.1 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 4 obstacles in the path. For the two agents problem, two conventional RL methods; Q-learning and SARSA and four hybrid-RL methods; RL with ACO, Phe-Q and Phe-SARSA and RL with PSO; Q-Swarm and SARSA-Swarm have been simulated.

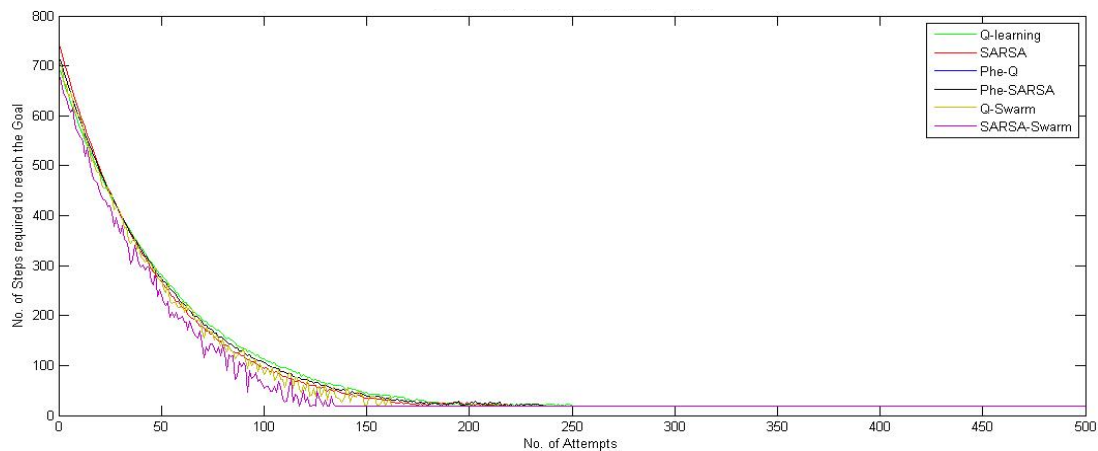


Figure 6.2.1 Plot between no. of steps required to reach the goal and no. of attempts for 2 agents; case I(a)

Figure 6.2.2a, 6.2.2b, 6.2.2c, 6.2.2d, 6.2.2e, 6.2.2f shows the grid world with the path traced by the agents using Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm, respectively for four obstacles placed the path. Blue and green lines show the paths traced by agent 1 and agent 2, respectively.

Here, the agents are very much free to move around and the obstacles are fewer as compared to the cells in which it can move. Thus, for all the six algorithms; the shortest path traced has the minimum number of steps as 19, but SARSA-Swarm reaches the optimal path for the minimum number of steps. From the above results, it can be seen that

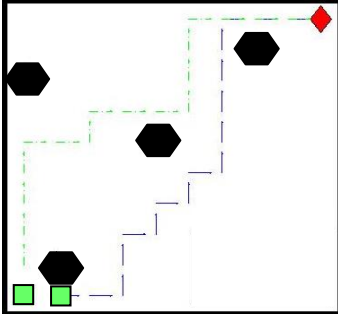


Figure 6.2.2a Path traced for case I(a) by two agents for Q-learning

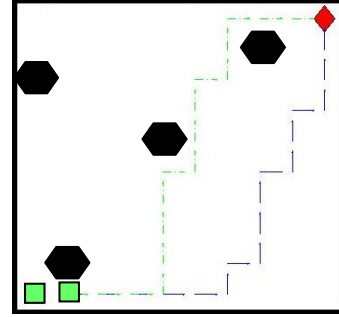


Figure 6.2.2b Path traced for case I(a) by two agents for SARSA

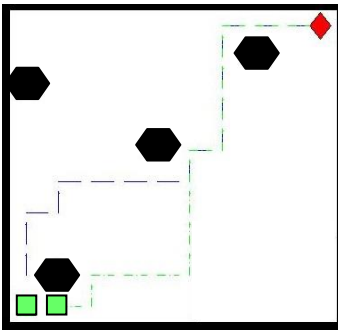


Figure 6.2.2c Path traced for case I(a) by two agents for Phe-Q

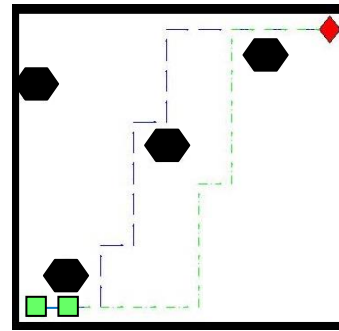


Figure 6.2.2d Path traced for case I(a) by two agents for Phe-SARSA

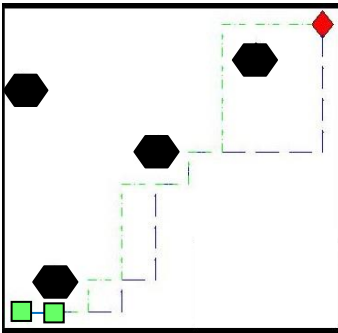


Figure 6.2.2e Path traced for case I(a) by single agent for Q-Swarm

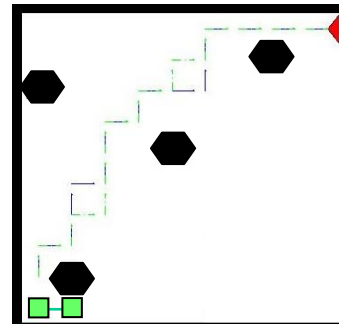


Figure 6.2.2f Path traced for case I(a) by single agent for SARSA-Swarm

for the Q-learning and SARSA, both the agents find path independently and thus the paths found are not near to each other as there is no coordination between the agents, and hence also takes more time to reach the optimal path. For the Phe-Q and Phe-SARSA, there is a kind of indirect interaction between the agents, as the pheromone trail left by an agent will be used as a guidance for both the agents and thus near the starting locations and the goal



location, both the agents follow the same path. For the Q-Swarm and SARSA-Swarm as the coordination between the agents is present, hence for these methods the optimal path was found for least number of attempts. Also, it can be seen that agents move in a swarm and are quite close to each other while reaching the goal.

(b) No. of obstacles = 8

Figure 6.2.3 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 8 obstacles in the path. Two conventional RL methods; Q-learning and SARSA and four hybrid-RL methods, i.e. RL with ACO; Phe-Q and Phe-SARSA and RL with PSO; Q-Swarm and SARSA-Swarm have been simulated.

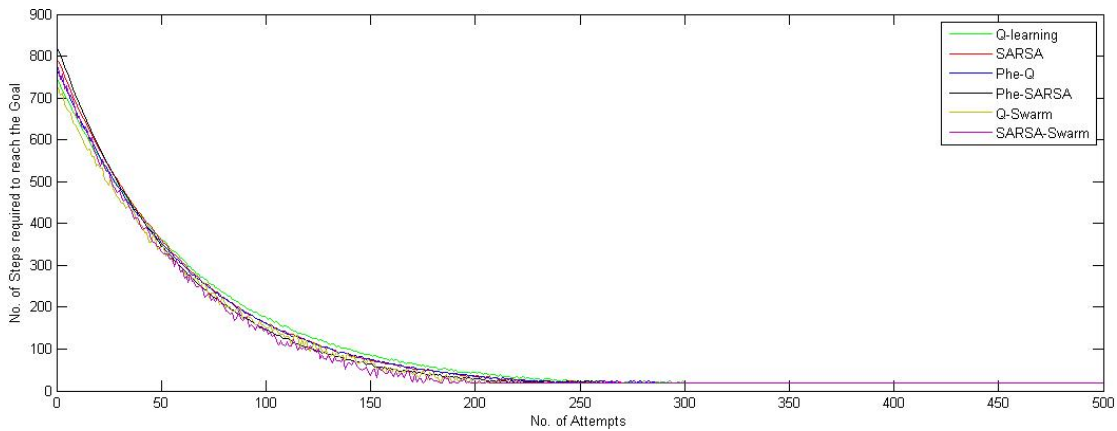


Figure 6.2.3 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case I(b)

Figure 6.2.4a, 6.2.4b, 6.2.4c, 6.2.4d, 6.2.4e, 6.2.4f shows the grid world with the path traced by the agents using Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm, respectively for eight obstacles placed the path.

Here also the agents are very free to move around and the obstacles are few as compared to the cells in which it can move. Thus, for all the six algorithms; the shortest path traced has the minimum number of steps as 19, but SARSA-Swarm reaches the optimal path for the minimum number of steps. Similar results have been obtained here as the previous one with 4 obstacles in the path. But with the increase in the number of obstacles in the path, there are more restrictions for the agents to move around and hence it takes more attempts for the agents to reach the optimal path as compared to the previous case.

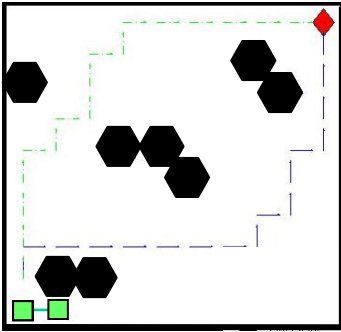


Figure 6.2.4a Path traced for case I(b) by two agents for Q-Learning

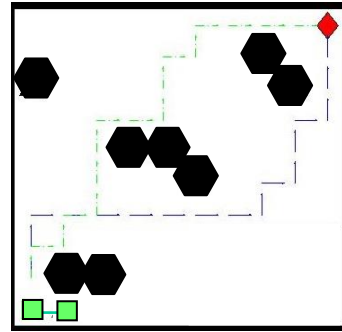


Figure 6.2.4b Path traced for case I(b) by two agents for SARSA

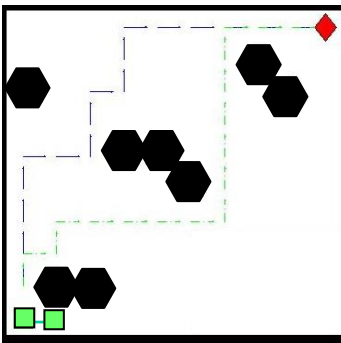


Figure 6.2.4c Path traced for case I(b) by two agents for Phe-Q

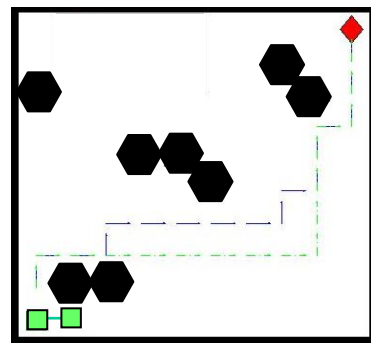


Figure 6.2.4d Path traced for case I(b) by two agents for Phe-SARSA

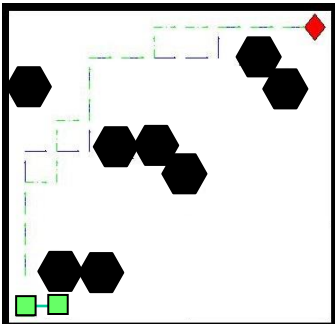


Figure 6.2.4e Path traced for case I(b) by two agents for Q-Swarm

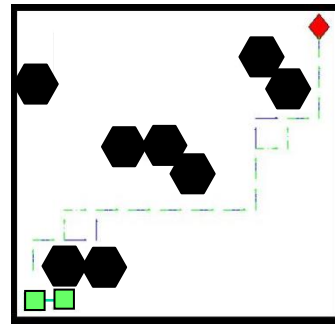


Figure 6.2.4f Path traced for case I(b) by two agents for SARSA-Swarm

(c) No. of obstacles = 14

Figure 6.2.5 shows the same case with more obstacles in the path. Simulations have been done for Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm for 14 obstacles placed in the path to reach goal.

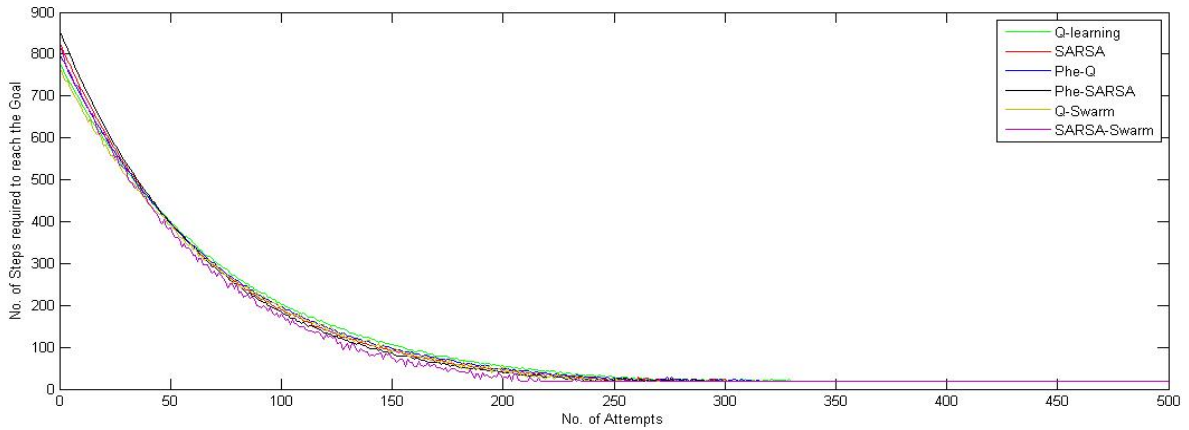


Figure 6.2.5 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case I(c)

Figure 6.2.6a, 6.2.6b, 6.2.6c, 6.2.6d, 6.2.6e, 6.2.6f shows the grid world with the path traced by the agents using Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm, respectively for fifteen obstacles placed the path.

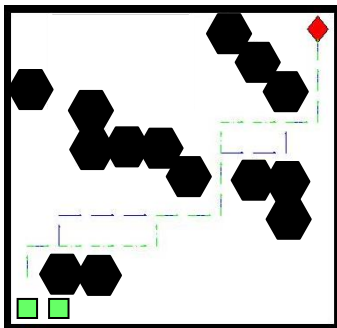


Figure 6.2.6a Path traced for case I(c) by two agents for Q-Learning

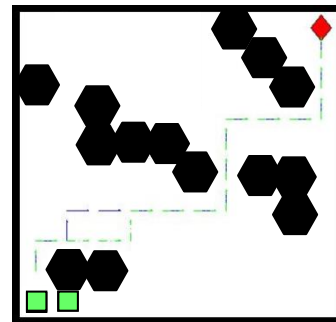


Figure 6.2.6b Path traced for case I(c) by two agents for SARSA

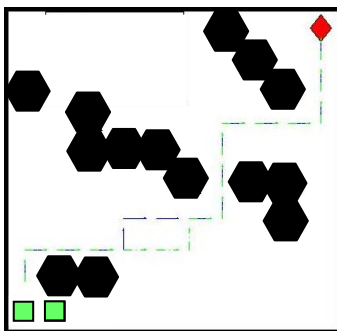


Figure 6.2.6c Path traced for case I(c) by two agents for Phe-Q

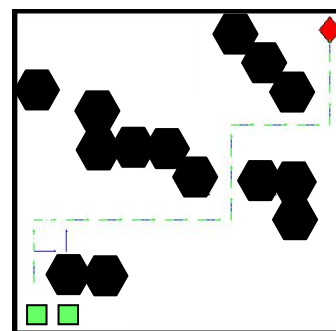


Figure 6.2.6d Path traced for case I(c) by two agents for Phe-SARSA

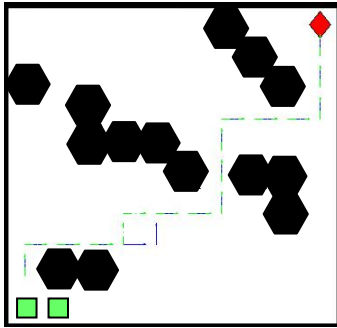


Figure 6.2.6e Path traced for case I(c) by two agents for Q-Swarm

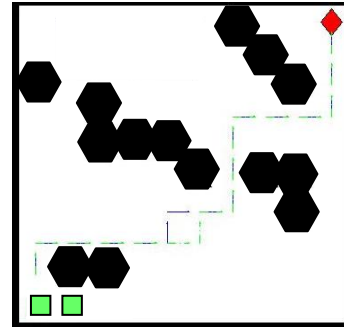


Figure 6.2.6f Path traced for case I(c) by two agents for SARSA-Swarm

Here, the agents are not free to move around and the obstacles are quite in number as compared to the cells in which it can move. For all the four algorithms; the shortest path traced has the minimum number of steps as 19, but SARSA-Swarm reaches the optimal path for the minimum number of steps. Here, it can be seen that the path taken the agents are similar in nature, this is so because there are only two paths for which the optimal number steps could be possible. With the increase in the number of obstacles in the path, there are more restrictions for the agents to move around and hence it takes more attempts for the agents to reach the optimal path as compared to the previous case.

## 6.2.2 Case II: Obstacles Fixed & Moving (Both), Goal Fixed

(a) No. of fixed obstacles = 4; No. of moving obstacles = 2

Figure 6.2.7 shows the no. of steps required by the agents to reach the goal against the no.

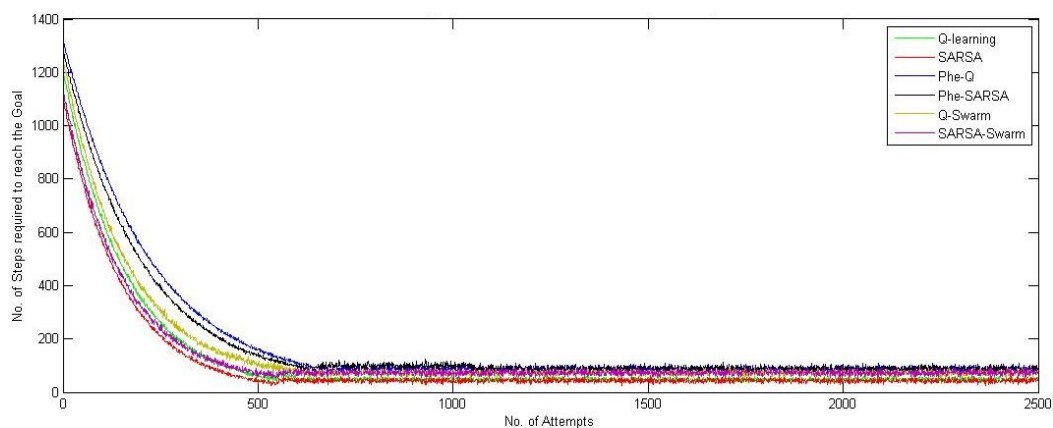


Figure 6.2.7 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case II(a)

of attempts for a 10X10 grid world with 4 fixed and 2 moving obstacles in the path. For this case six methods have been simulated: two conventional RL methods; Q-learning and SARSA, and four hybrid-RL methods: RL with ACO; Phe-Q and Phe-SARSA and RL with Swarm; Q-Swarm and SARSA- Swarm.

Figure 6.2.8a, 6.2.8b shows the grid world with the path traced by the agents using Q-learning and SARSA, respectively for 4 fixed and two moving obstacles placed in the path. The fixed obstacles are shown by black hexagon and the moving ones by blue. The cells in which the moving obstacles can move are shown using the grey patch.

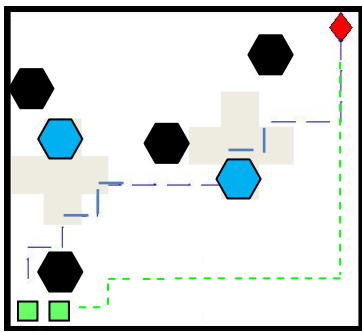


Figure 6.2.8a Path traced for case II(a) by two agents for Q-learning

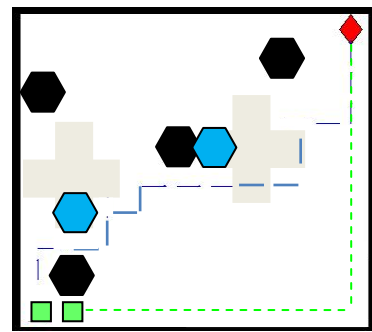


Figure 6.2.8b Path traced for case II(a) by two agents for SARSA

The result shows that for such a case the Q-learning and SARSA find the optimal path faster as compared to the hybrid ones. Q-swarm and SARSA-swarm still simulates to reach to an optimal path but Phe-Q and Phe-SARSA are not able to find this optimal path. The optimal steps value obtained in these methods is higher than the rest of the four methods. This is so because the Phe-Q and Phe-SARSA uses the pheromone level of each grid into account and with moving obstacles in the path, the pheromone level might misguide the agent to take a path which has does not have obstacles for current simulation but might have for the next one. Whereas in the swarm methods, they agents work collectively to reach the goal and in the process of reaching the goal do not alter the environment as with the pheromone algorithms do.

(b) No. of fixed obstacles = 5; No. of moving obstacles = 3

Figure 6.2.9 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 5 fixed and 3 moving obstacles in the path. For this

case six methods have been simulated: two conventional RL methods; Q-learning and SARSA, and four hybrid-RL methods: RL with ACO; Phe-Q and Phe-SARSA and RL with Swarm; Q-Swarm and SARSA- Swarm.

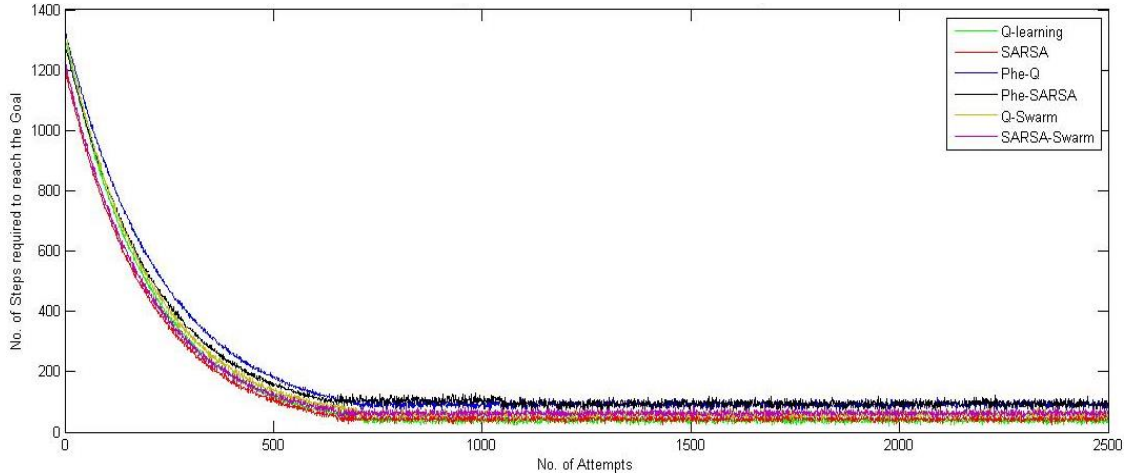


Figure 6.2.9 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case II(b)

Figure 6.2.10a, 6.2.10b, 6.2.10c, 6.2.10d, shows the grid world with the path traced by the agents using Q-learning, SARSA, Q-Swarm and SARSA-Swarm, respectively for 5 fixed and 3 moving obstacles placed in the path. The fixed obstacles are shown by black hexagons and the moving ones by blue. The cells in which the moving obstacles can move are shown using the grey patch.

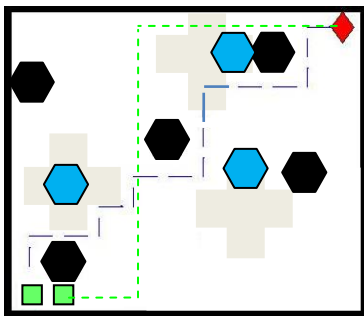


Figure 6.2.10a Path traced for case II(b) by two agents for Q-learning

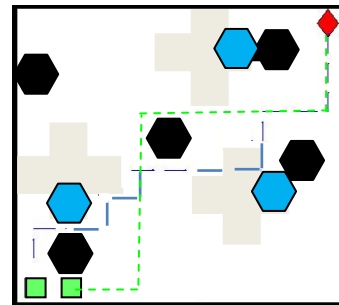


Figure 6.2.10b Path traced for case II(b) by two agents for SARSA

The results obtained are similar to the previous case. Though agent could reach the goal for all the methods, Q-learning and SARSA find the optimal path faster as compared to the

hybrid ones. Q-swarm and SARSA-swarm still simulates to reach to an optimal path but Phe-Q and Phe-SARSA are not able to find this optimal path. The optimal steps value obtained in these methods is higher than the rest of the four methods. Also, on increasing the number of fixed and moving obstacles, the optimal value reached by Phe-Q and Phe-SARSA are very high than the rest of the four methods.

### 6.2.3 Case III: Obstacles Fixed, Goal Moving

(a) No. of fixed obstacles = 4

Figure 6.2.11 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 4 obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm.

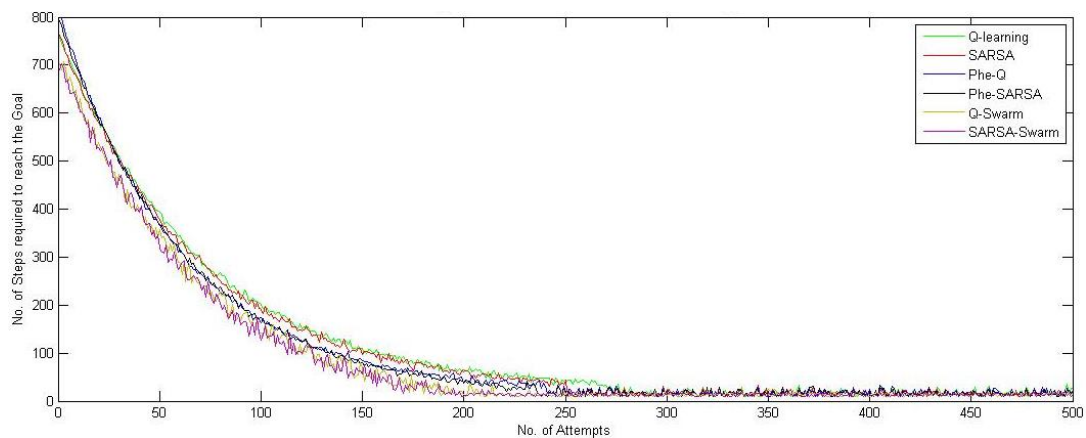


Figure 6.2.11 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case III(a)

Figure 6.2.12a, 6.2.12b, 6.2.12c, 6.2.12d, 6.2.12e, 6.2.12f shows the grid world with the path traced by the agents when the goal is moving and there are 4 obstacles in the path for Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm.

Here, for all the six methods the agents are able to find to optimal paths after various number of attempts. But, SARSA-swarm method is able to find the shortest path for any given goal loactions in the least number of attempts. This is so because, SARSA-swarm or Q-swarm, the agents coordinate with each other to tell the possible locations of the goal.

The global maxima is thus here not for a particular cell, but for all the cells for which goal could possibly be present. As the number of attempts increases, the agents have an understanding of moving vertically to reach the bands where goal can be present and then traverse to look for the actual position of the goal.

The moving goal area is represented by the grey covered region.

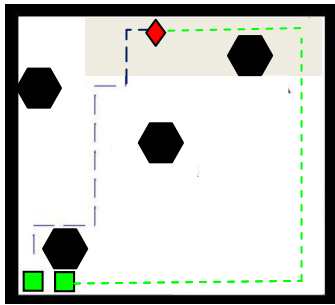


Figure 6.2.12a Path traced for case III(a) by two agents for Q-learning

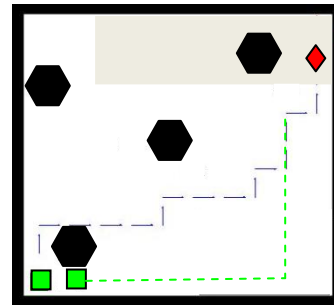


Figure 6.2.12b Path traced for case III(a) by two agents for SARSA

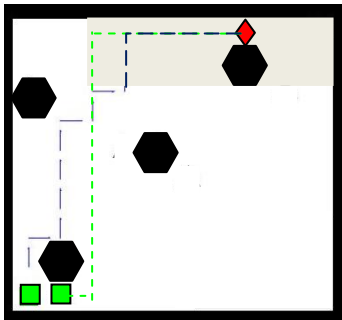


Figure 6.2.12c Path traced for case III(a) by two agents for Phe-Q

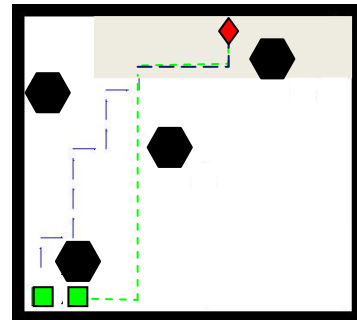


Figure 6.2.12d Path traced for case III(a) by two agents for Phe-SARSA

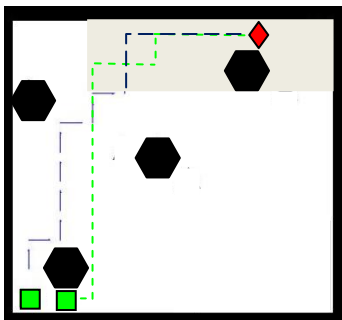


Figure 6.2.12c Path traced for case III(a) by two agents for Q-Swarm

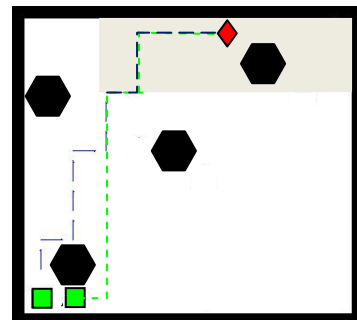


Figure 6.2.12d Path traced for case III(a) by two agents for SARSA-Swarm



(b) No. of fixed obstacles = 8

Figure 6.2.13 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 8 obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA, Q-Swarm and SARSA-Swarm.

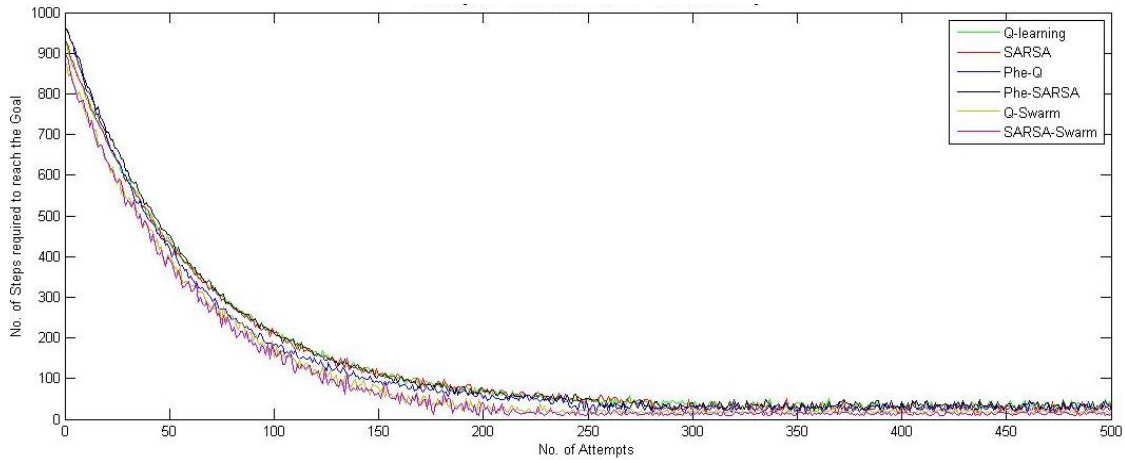


Figure 6.2.13 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case III(b)

Figure 6.2.14a, 6.2.14b, 6.2.14c, 6.2.14d, 6.2.14e, 6.2.14f shows the grid world with the path traced by the agents when the goal is moving and there are 8 obstacles in the path for Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm.

The moving goal area is represented by the grey covered region.

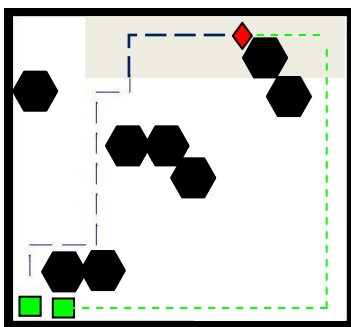


Figure 6.2.14a Path traced for case III(b) by two agents for Q-learning

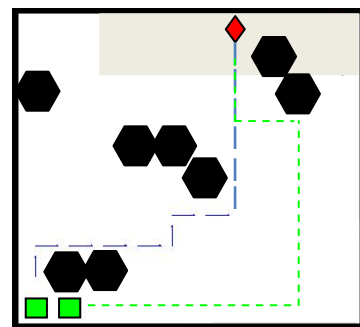


Figure 6.2.14b Path traced for case III(b) by two agents for SARSA

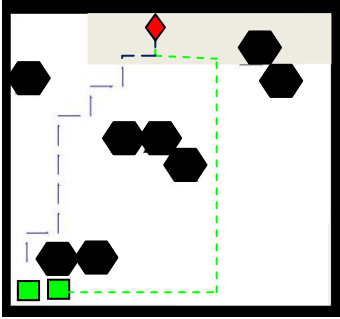


Figure 6.2.14c Path traced for case III(b)  
by two agents for Phe-Q

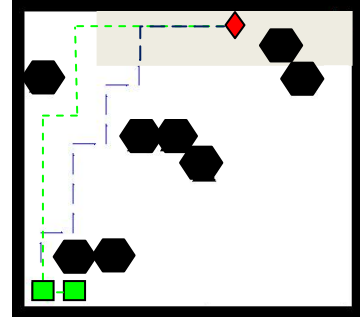


Figure 6.2.14d Path traced for case III(b)  
by two agents for Phe-SARSA

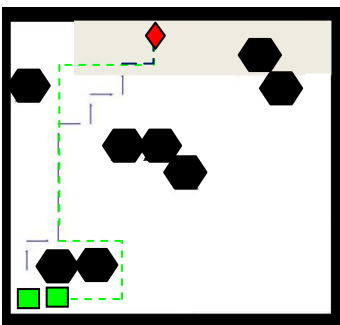


Figure 6.2.14e Path traced for case III(b)  
by two agents for Q-Swarm

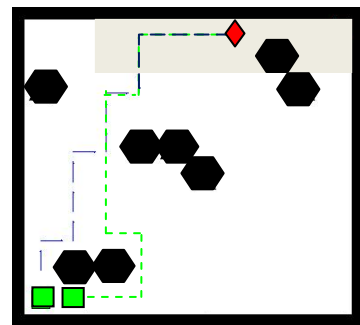


Figure 6.2.14f Path traced for case III(b)  
by two agents for SARSA-Swarm

Here also, for all the six methods the agents are able to find to optimal paths after various number of attempts. But, SARSA-swarm method is able to find the shortest path for any given goal loactions in the least number of attempts. On increasing the number of obstacles present in the path, it takes more attempts for all the methods to their own optimal values.

(c) No. of fixed obstacles = 14

Figure 6.2.15 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 14 obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: the two conventional reinforcement algorithms: Q-learning and SARSA, four hybrid algorithms, RL with ACO: Phe-Q and Phe-SARSA and RL with PSO: Q-Swarm and SARSA-Swarm.

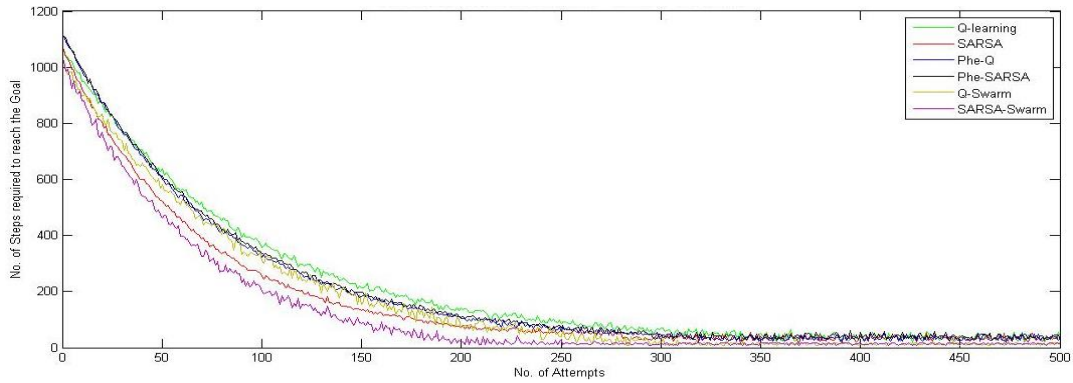


Figure 6.2.15 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case III(c)

Figure 6.2.16a, 6.2.16b, 6.2.16c, 6.2.16d, 6.2.16e, 6.2.16f shows the grid world with the path traced by the agents when the goal is moving and there are 8 obstacles in the path for Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm.

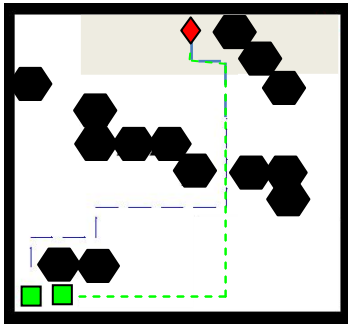


Figure 6.2.16a Path traced for case III(c) by two agents for Q-learning

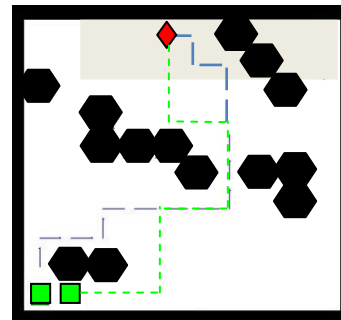


Figure 6.2.16b Path traced for case III(c) by two agents for SARSA

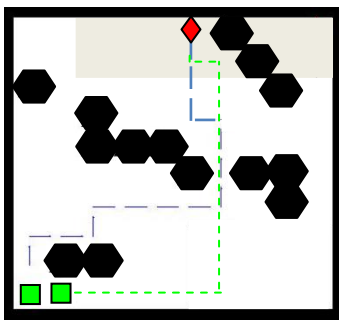


Figure 6.2.16c Path traced for case III(c) by two agents for Phe-Q

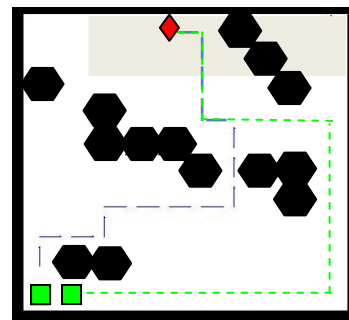


Figure 6.2.16d Path traced for case III(c) by two agents for Phe-SARSA

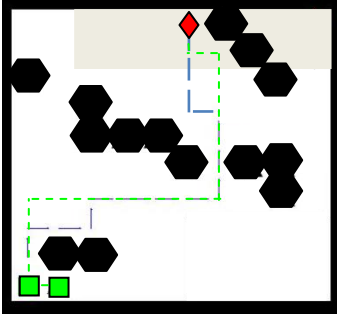


Figure 6.2.16e Path traced for case III(c) by two agents for Q-Swarm

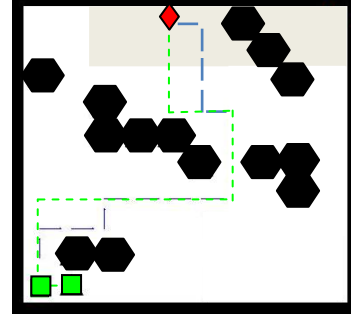


Figure 6.2.16f Path traced for case III(c) by two agents for SARSA-Swarm

Here also, for all the six methods the agents are able to find to optimal paths after various number of attempts. But, SARSA-swarm method is able to find the shortest path for any given goal loactions in the least number of attempts. With the increase in the number of obstacles present in the path, it takes more attempts for all the methods to their own optimal values. It was also observed that with the normal RL methods the agent may take any random path independent of what other agent is moving. For Phe-Q and Phe-SARSA methods, usually near the starting location and near the end location, agents take same path. This is because of the high pheromone levels. With the Q-Swarm and SARSA-Swarm methods, the agents try to follow each others steps and try to move in a connected group or swarm.

#### 6.2.4 Case IV: Obstacles Fixed & Moving (Both), Goal Moving

(a) No. of fixed obstacles = 4; No. of moving obstacles = 2

Figure 6.2.17 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 4 fixed and 2 moving obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA, Q-Swarm and SARSA-Swarm.

Comparing the six methods, as the system dynamics increases and the obstacles also started to move, SARSA-Swarm and Q-Swarm converges to obtain an optimal path, but the former converges first. Q-learning, SARSA, Phe-Q and Phe-SARSA also converge to find some solution but they do not reach to find the optimal solution in this case.

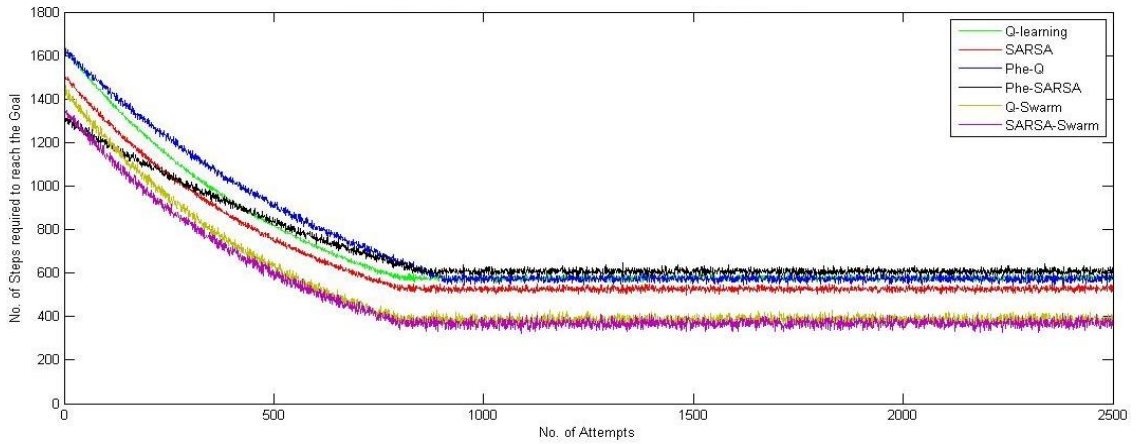


Figure 6.2.17 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case IV(a)

(b) No. of fixed obstacles = 5; No. of moving obstacles = 3

Figure 6.2.18 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 5 fixed and 3 moving obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA, Q-Swarm and SARSA-Swarm.

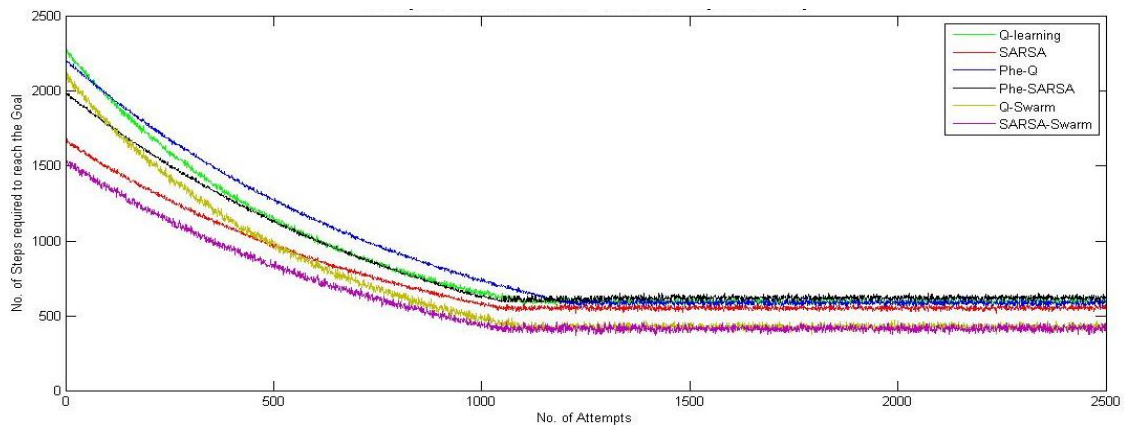


Figure 6.2.18 Plot between No. of Steps required to reach the Goal and No. of Attempts for 2 agents; case IV(b)

Comparing the six methods, as the system dynamics increases and the obstacles also started to move, SARSA-Swarm and Q-Swarm converges to obtain an optimal path, but the former converges first. Q-learning, SARSA, Phe-Q and Phe-SARSA also converge to find some solution but they do not reach to find the optimal solution in this case.

A comparative analysis for the computation time taken by all the six algorithms for the different cases simulated is provided in the table 6.2.

Table 6.2 Computational Time for the Two Agent Problem for the six algorithms simulated

CASE	SARSA	Q-Learning	Phe-Q	Phe-SARSA	Q-Swarm	SARSA-Swarm
I	0.004010837	0.009909973	0.00939	0.008636909	0.0075958	0.007551244
II	0.003813394	0.181120568	0.009282	0.009200768	0.0079911	0.007977043
III	0.003791831	0.035102582	0.009517	0.009345154	0.0079509	0.007866822
IV	0.016171792	0.032133351	0.034206	0.030490695	0.0149398	0.012478041
V	0.024881458	0.043875963	0.058414	0.051943521	0.0205003	0.019030869
VI	0.008152752	0.012631409	0.016528	0.016049484	0.0080581	0.007438657
VII	0.008687107	0.016438786	0.017385	0.016832651	0.0086036	0.00767714
VIII	0.006935863	0.011599468	0.013881	0.01363516	0.0068813	0.006838816
IX	0.015110023	0.026175395	0.03024	0.02719235	0.0135855	0.013153943
X	0.172205847	0.331158611	0.33233	0.329371827	0.1710423	0.169728648

The computation table shows that the SARSA computed fastest out of all the six algorithms used. Q-learning is slower than SARSA because for Q-learning for the Q updation formula, we use a max function which uses more memory for computation whereas in SARSA such a function is not required. The overall time computation for the Phe-Q or Phe-SARSA is around two times that of the conventional RL, this is so because for these methods the ant needs to reach the goal and come back again to the initial starting location before the next simulation is carried out. For the swarm algorithms, as the number of parameters that needs to be calculated increases, for each attempt run the velocity parameters, global and personal best Q values are calculated and also updated. This increases the overall time of simulation for the swarm algorithms.

### 6.3 Four Agents Problem

For the four agent case, following results was obtained for the various environmental cases:

#### 6.3.1 Case I: Obstacles Fixed, Goal Fixed

(a) No. of obstacles = 4

Figure 6.3.1 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10 X 10 grid world with 4 obstacles in the path. For the four agents

problem, two conventional RL methods; Q-learning and SARSA and four hybrid-RL methods; RL with ACO, Phe-Q and Phe-SARSA and RL with PSO; Q-Swarm and SARSA-Swarm have been simulated.

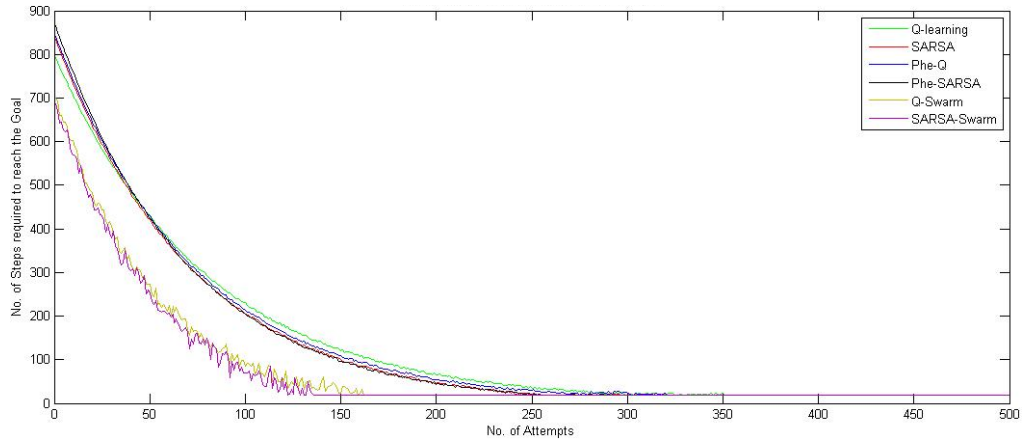


Figure 6.3.1 Plot between no. of steps required to reach the goal and no. of attempts for 4 agents; case I(a)

Figure 6.3.2a, 6.3.2b, 6.3.2c, 6.3.2d, 6.3.2e, 6.3.2f shows the grid world with the path traced by the agents using Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm, respectively for four obstacles placed the path. For the four agents, the starting point is made different. They all start at different but nearby loactions to reach the common goal.

Green, red, black and blue lines paths traced by agent 1, agent 2, agent 3 and agent 4, respectively.

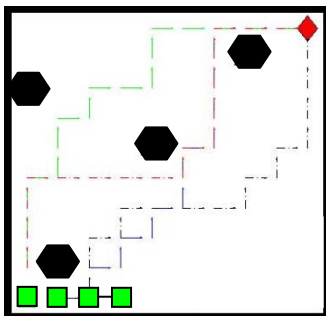


Figure 6.3.2a Path traced for case I(a) by four agents for Q-learning

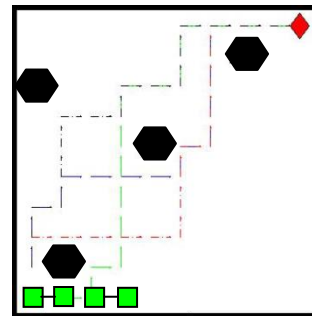


Figure 6.3.2b Path traced for case I(a) by four agents for SARSA

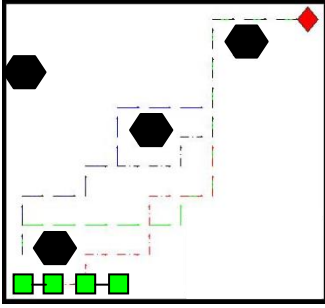


Figure 6.3.2c Path traced for case I(a)  
by four agents for Phe-Q

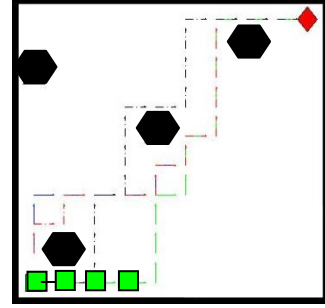


Figure 6.3.2d Path traced for case I(a)  
by four agents for Phe-SARSA

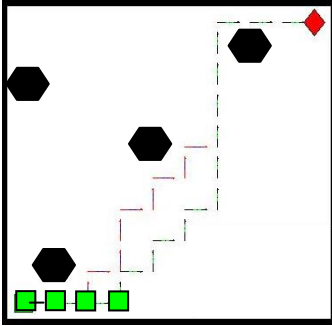


Figure 6.3.2e Path traced for case I(a)  
by four agents for Q-Swarm

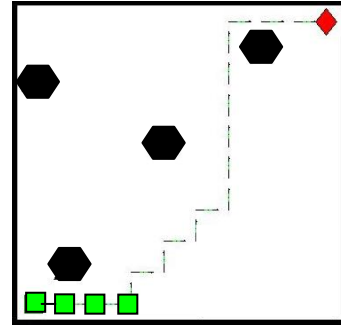


Figure 6.3.2f Path traced for case I(a)  
by four agents for SARSA-Swarm

Here, the agents are very much free to move around and the obstacles are fewer as compared to the cells in which it can move. Thus, for all the four algorithms; the shortest path traced has the minimum number of steps as 22, but SARSA-Swarm reaches the optimal path for the minimum number of steps. From the above results, it can be seen that for the Q-learning and SARSA, both the agents find path independently and thus the paths found are not near to each other as there is no coordination between the agents, and hence also takes more time to reach the optimal path. For the Phe-Q and Phe-SARSA, there is a kind of indirect interaction between the agents, as the pheromone trail left by an agent will be used as a guidance for both the agents and thus near the starting locations and the goal location, both the agents follow the same path. For the Q-Swarm and SARSA-Swarm as the coordination between the agents is present, hence for these methods the optimal path was found for least number of attempts. Also, it can be seen that agents move in a swarm and are quite close to each other while reaching the goal.



(b) No. of obstacles = 8

Figure 6.3.3 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 8 obstacles in the path. Two conventional RL methods, Q-learning and SARSA and two hybrid-RL methods, i.e. RL with ACO, Phe-Q and Phe-SARSA have been simulated.

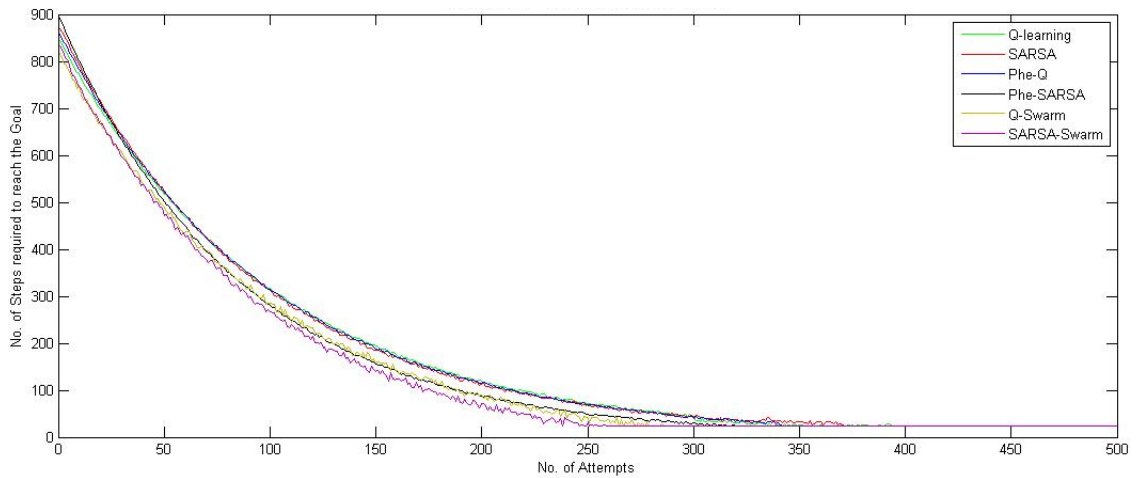


Figure 6.3.3 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case I(b)

Figure 6.3.4a, 6.3.4b, 6.3.4c, 6.3.4d, 6.3.4e, 6.3.4f shows the grid world with the path traced by the agents using Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm, respectively for eight obstacles placed the path. Different coloured lines indicate the path travelled by the different agents.

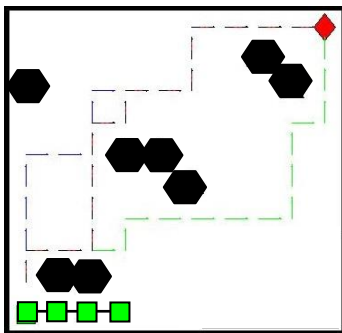


Figure 6.3.4a Path traced for case I(b) by four agents for Q-Learning

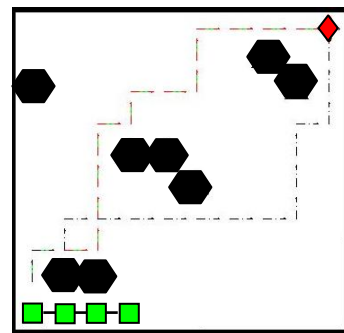


Figure 6.3.4b Path traced for case I(b) by four agents for SARSA

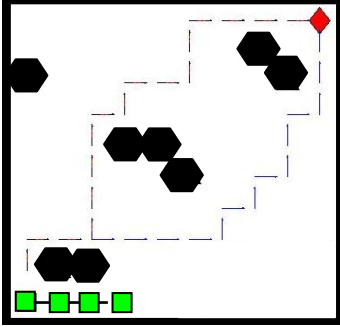


Figure 6.3.4c Path traced for case I(b) by four agents for Phe-Q

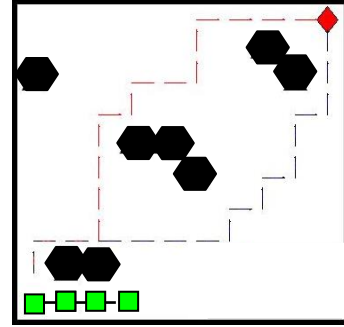


Figure 6.3.4d Path traced for case I(b) by four agents for Phe-SARSA

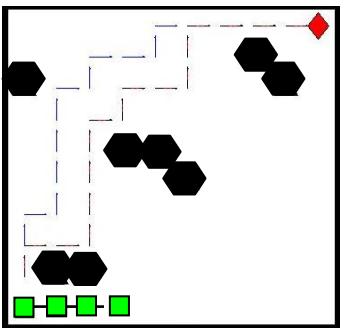


Figure 6.3.4e Path traced for case I(b) by four agents for Q-Swarm

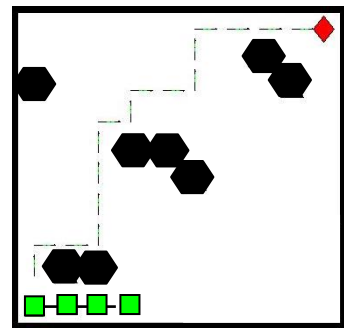


Figure 6.3.4f Path traced for case I(b) by four agents for SARSA-Swarm

Here also the agents are very free to move around and the obstacles are few as compared to the cells in which it can move. Thus, for all the six algorithms; the shortest path traced has the minimum number of steps as 22, but SARSA-swarm reaches the optimal path for the minimum number of steps. Similar results have been obtained here as the previous one with 4 obstacles in the path. But with the increase in the number of obstacles in the path, there are more restrictions for the agents to move around and hence it takes more attempts for the agents to reach the optimal path as compared to the previous case.

(c) No. of obstacles = 14

Figure 6.3.5 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 14 obstacles in the path. Two conventional RL methods, Q-learning and SARSA and two hybrid-RL methods, i.e. RL with ACO, Phe-Q and Phe-SARSA have been simulated.

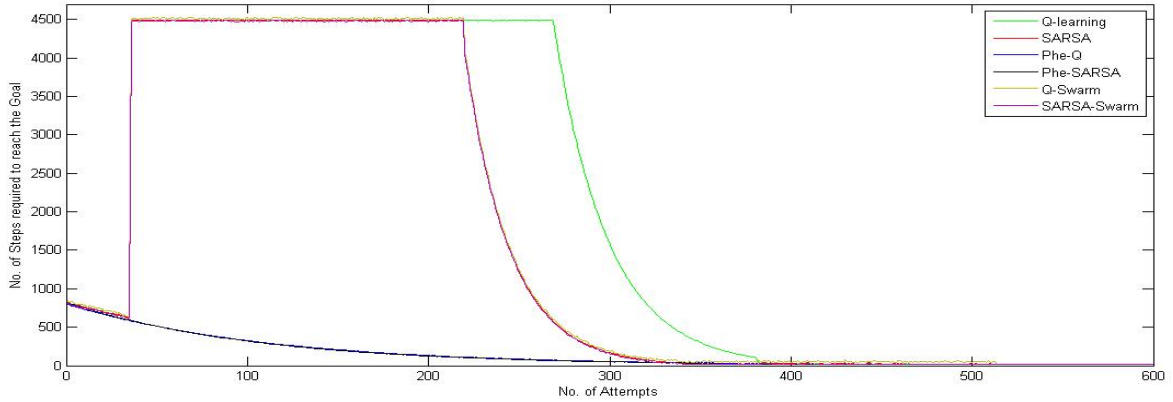


Figure 6.3.5 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case I(c)

Figure 6.3.6a, 6.3.6b, 6.3.6c, 6.3.6d, 6.3.6e, 6.3.6f shows the grid world with the path traced by the agents using Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm, respectively for fifteen obstacles placed the path.

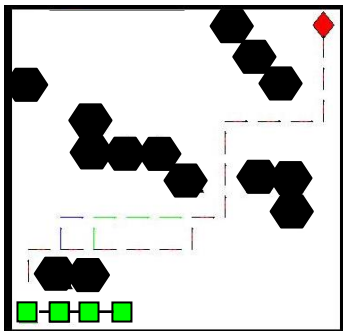


Figure 6.3.6a Path traced for case I(c) by four agents for Q-Learning

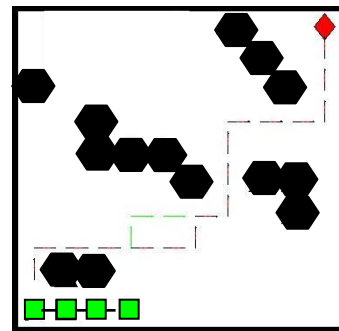


Figure 6.3.6b Path traced for case I(c) by four agents for SARSA

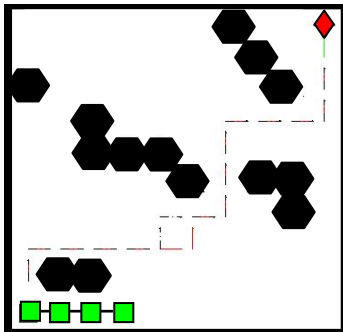


Figure 6.3.6c Path traced for case I(c) by four agents for Phe-Q

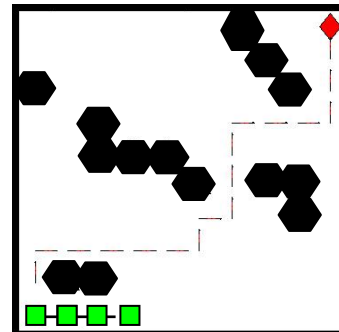


Figure 6.3.6d Path traced for case I(c) by four agents for Phe-SARSA

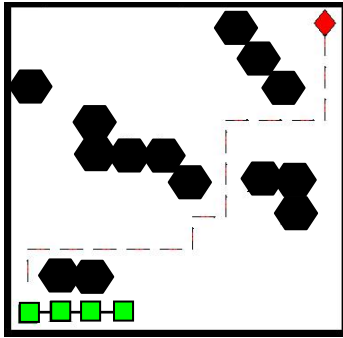


Figure 6.3.6e Path traced for case I(c) by four agents for Q-Swarm

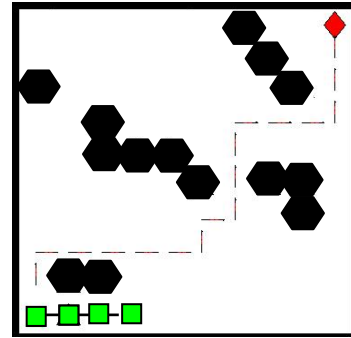


Figure 6.3.6f Path traced for case I(c) by four agents for SARSA-Swarm

Here, the agents are not free to move around and the obstacles are quite in number as compared to the cells in which it can move. For all the six algorithms; the shortest path traced has the minimum number of steps as 22, but SARSA-Swarm reaches the optimal path for the minimum number of steps. Here, it can be seen that the path taken the agents are similar in nature, this is so because there are only two paths for which the optimal number steps could be possible. With the increase in the number of obstacles in the path, there are more restrictions for the agents to move around and hence it takes more attempts for the agents to reach the optimal path as compared to the previous case.

### 6.3.2 Case II: Obstacles Fixed & Moving (Both), Goal Fixed

(a) No. of fixed obstacles = 4; No. of moving obstacles = 2

Figure 6.3.7 shows the no. of steps required by the agents to reach the goal against the no.

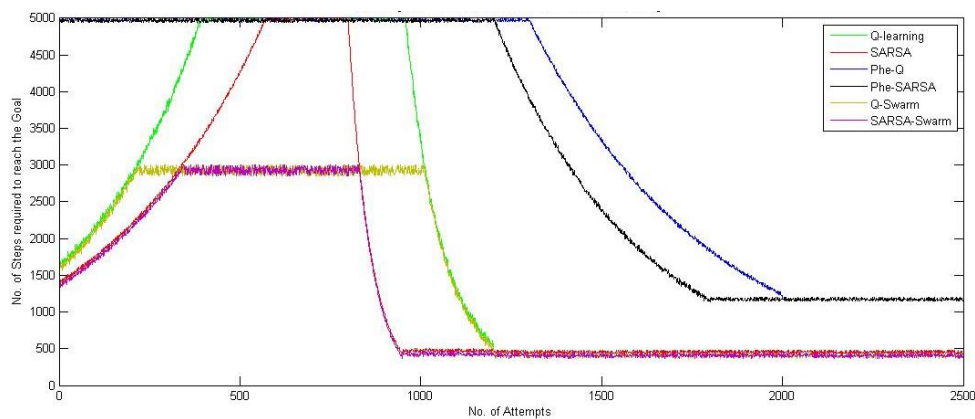


Figure 6.3.7 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case II(a)

of attempts for a 10X10 grid world with 4 fixed and 2 moving obstacles in the path. For this case six methods have been simulated: two conventional RL methods; Q-learning and SARSA, and four hybrid-RL methods: RL with ACO; Phe-Q and Phe-SARSA and RL with Swarm; Q-Swarm and SARSA- Swarm.

The result shows that for such a case the SARSA-swarm finds the optimal path for the least number of attempts as compared to the other four algorithms. Also, the Q-learning and the SARSA methods do converge to this optimal number of steps but takes more attempts to reach this optimal solution. This is so because in the Q-swarm and SARSA-swarm the agents coordinate with each other and the Q values updated uses both the individual best performance and also the best performance among the group. As compared to these methods, the Phe methods do not converge to the optimal values of steps, this is so because in the Phe methods the ants lay pheromone to the various cells they travelled and due to moving obstacles in the path, this pheromone level distributed is quite even and this causes confusion for the agents to travel to the next cell.

(b) No. of fixed obstacles = 4; No. of moving obstacles = 8

Figure 6.3.8 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 4 fixed and 8 moving obstacles in the path. For this case six methods have been simulated: two conventional RL methods; Q-learning and SARSA, and four hybrid-RL methods: RL with ACO; Phe-Q and Phe-SARSA and RL with Swarm; Q-Swarm and SARSA- Swarm.

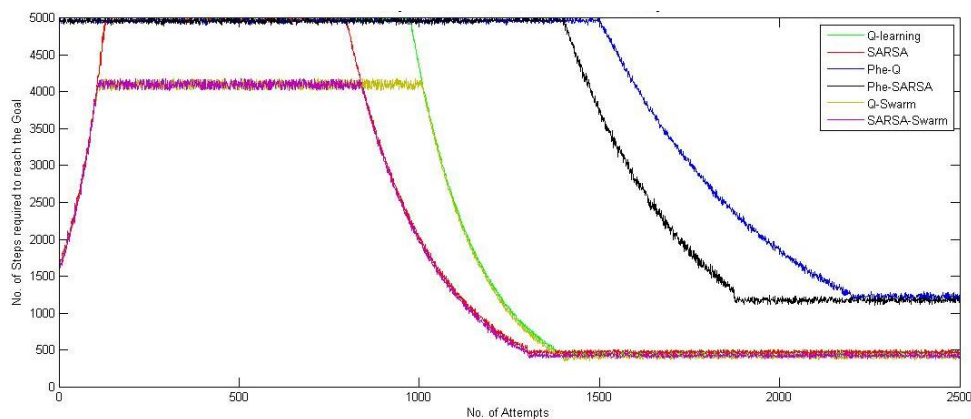


Figure 6.3.8 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case II(b)

The result obtained here is similar to the previous case, and here also SARSA-swarm finds the optimal path for the least number of attempts as compared to the other four algorithms. Also, the Q-learning and the SARSA methods do converge to this optimal number of steps but takes more attempts to reach this optimal solution. As compared to these methods, the Phe methods do not converge to the optimal values of steps. With increase in the number of obstacles in the path both stationary and the moving ones the overall steps required to reach the goal increases. This difference between steps for swarm methods and Phe methods also increases.

### 6.2.3 Case III: Obstacles Fixed, Goal Moving

(a) No. of fixed obstacles = 4

Figure 6.3.9 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 4 obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA, Q-Swarm and SARSA-Swarm.

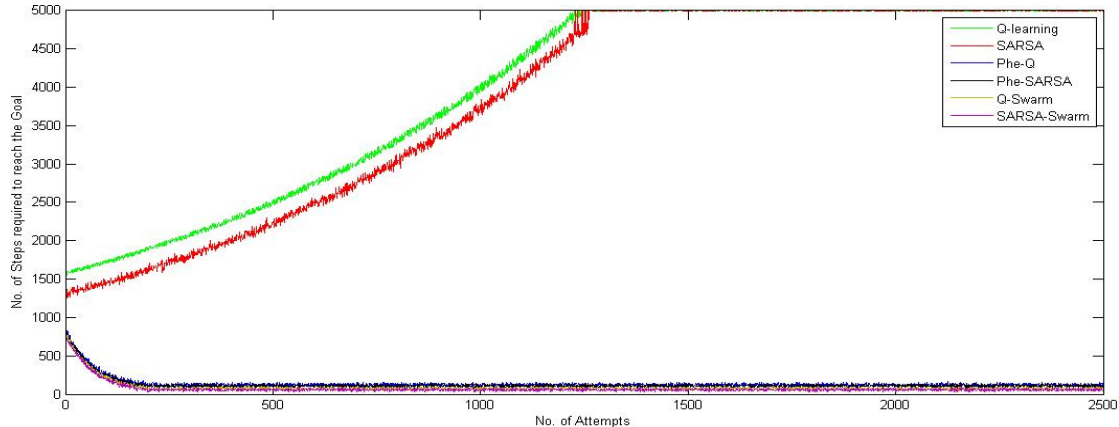


Figure 6.3.9 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case III(a)

Figure 6.3.10a, 6.3.10b, 6.3.10c, 6.3.10d shows the grid world with the path traced by the agents when the goal is moving and there are 4 obstacles in the path for Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm, respectively.

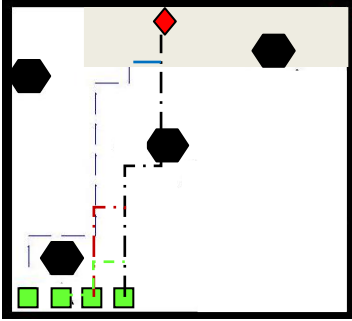


Figure 6.3.10a Path traced for case III(a) by four agents for Phe-Q

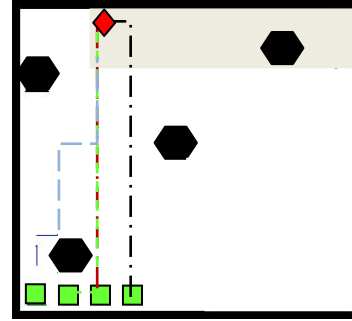


Figure 6.3.10b Path traced for case III(a) by four agents for Phe-SARSA

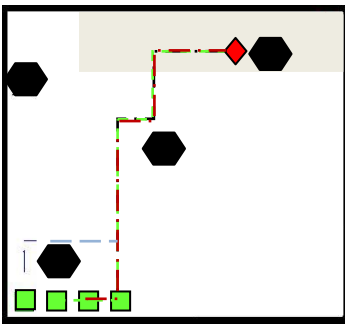


Figure 6.3.10c Path traced for case III(a) by four agents for Q-Swarm

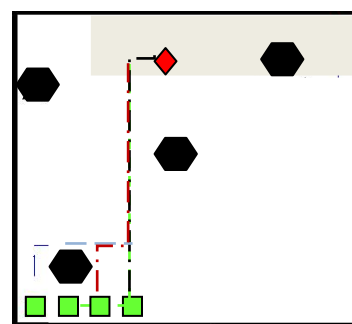


Figure 6.3.10d Path traced for case III(a) by two four for SARSA-Swarm

Here, for all the six methods, conventional RL methods are not able to find any optimal paths and they diverge and reach the maximum steps per attempt criterion for termination. Since, there is no coordination between the agents so they are not able to reach to a common goal. The other four methods are able to find the optimal path. But, SARSA-swarm method is able to find the shortest path for any given goal locations in the least number of attempts. This is so because, SARSA-swarm or Q-swarm, the agents coordinate with each other to tell the possible locations of the goal. The global maxima is thus here not for a particular cell, but for all the cells for which goal could possibly be present. As the number of attempts increases, the agents have an understanding of moving vertically to reach the bands where goal can be present and then traverse to look for the actual position of the goal. Phe methods also find the optimal paths for the given case. It can be seen that mostly, the agents follow the same paths for Phe methods when near to the starting location or goal. This is so because the pheromone level is concentrated for few cells only in these

regions but for other cells they a little bit distributed.

(b) No. of fixed obstacles = 8

Figure 6.3.11 shows the no. of steps required by the agents to reach the goal against the no. of attempts for a 10X10 grid world with 8 obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA, Q-Swarm and SARSA-Swarm.

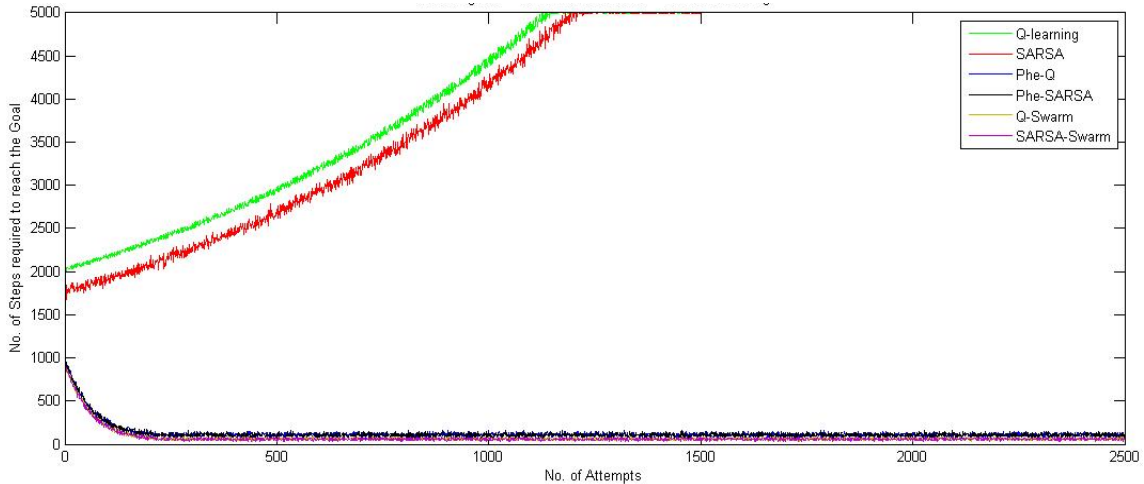


Figure 6.3.11 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case III(b)

Figure 6.3.12a, 6.3.12b, 6.3.12c, 6.3.1d shows the grid world with the path traced by the agents when the goal is moving and there are 8 obstacles in the path for Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm.

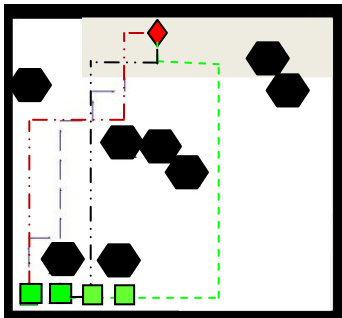


Figure 6.3.12a Path traced for case III(b) by four agents for Phe-Q

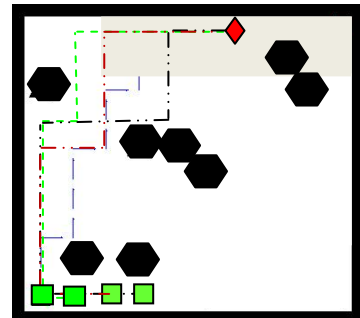


Figure 6.3.12b Path traced for case III(b) by four agents for Phe-SARSA



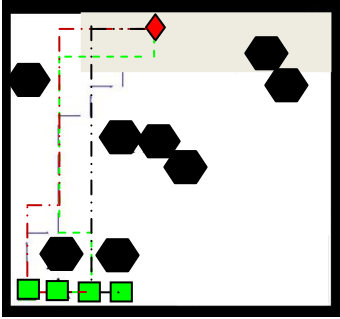


Figure 6.3.12c Path traced for case III(b) by four agents for Q-Swarm

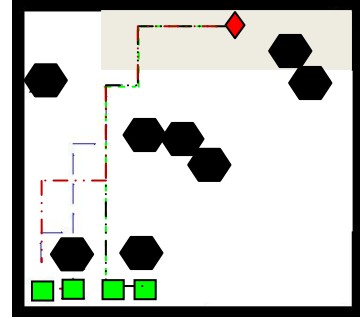


Figure 6.3.12d Path traced for case III(b) by four agents for SARSA-Swarm

Here also for all the six methods, conventional RL methods are not able to find any optimal paths and they diverge and reach the maximum steps per attempt criterion for termination. Since, there is no coordination between the agents so they are not able to reach to a common goal. The other four methods are able to find the optimal path. But, SARSA-swarm method is able to find the shortest path for any given goal locations in the least number of attempts. With the increase in the number of obstacles in the path, the convergence to find the optimal path requires more number of attempts. This happens because as the number of obstacles increases, the agents are not able to explore the grid world that efficiently.

(c) No. of fixed obstacles = 14

Figure 6.3.13 shows the no. of steps required by the agents to reach the goal against the no.

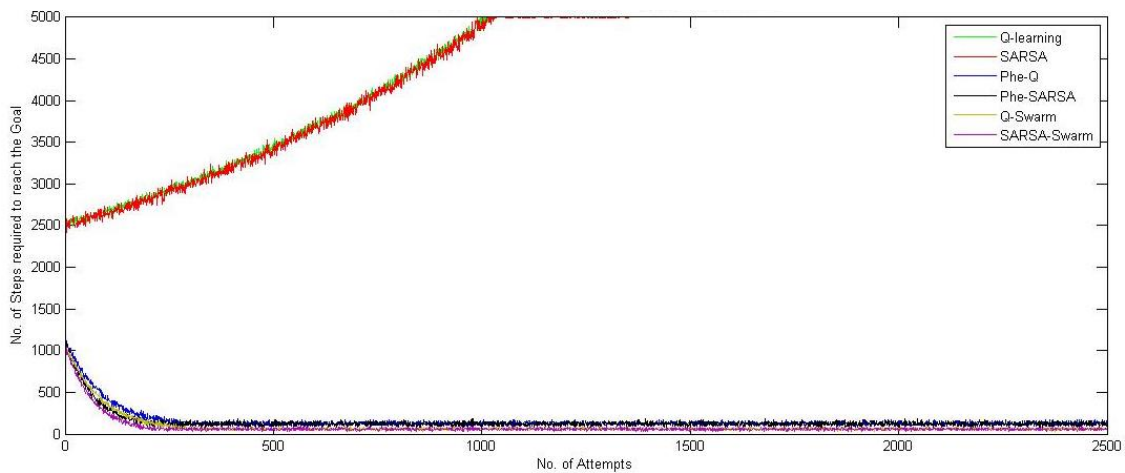


Figure 6.3.13 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case III(c)

of attempts for a 10X10 grid world with 14 obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA, Q-Swarm and SARSA-Swarm.

Figure 6.3.14a, 6.3.14b, 6.3.14c, 6.3.14d shows the grid world with the path traced by the agents when the goal is moving and there are 15 obstacles in the path for Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm.

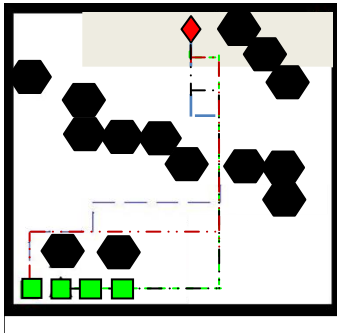


Figure 6.3.14a Path traced for case III(c) by four agents for Phe-Q

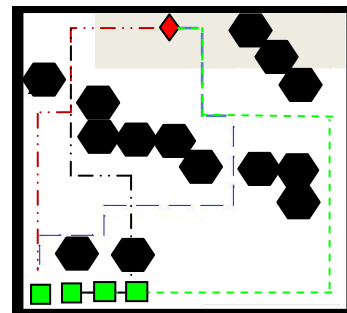


Figure 6.3.14d Path traced for case III(c) by four agents for Phe-SARSA

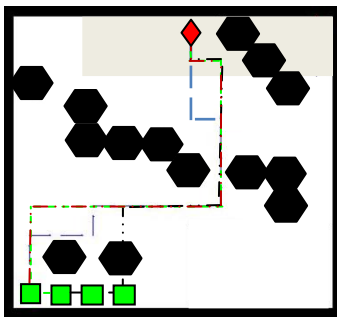


Figure 6.3.14c Path traced for case III(c) by four agents for Q-Swarm

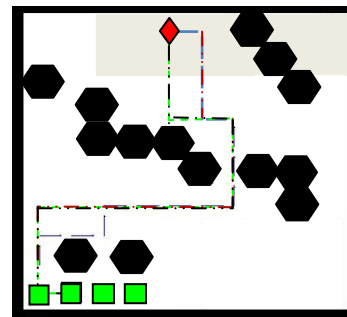


Figure 6.3.14f Path traced for case III(c) by four agents for SARSA-Swarm

Here also for all the six methods, conventional RL methods are not able to find any optimal paths and they diverge and reach the maximum steps per attempt criterion for termination. Since, there is no coordination between the agents so they are not able to reach to a common goal. The other four methods are able to find the optimal path. But, SARSA-swarm method is able to find the shortest path for any given goal locations in the least number of attempts. With the increase in the number of obstacles in the path, the convergence to find the optimal path requires more number of attempts. This happens

because as the number of obstacles increases, the agents are not able to explore the grid world that efficiently.

### 6.3.4 Case IV: Obstacles Fixed & Moving (Both), Goal Moving

(a) No. of fixed obstacles = 4; No. of moving obstacles = 2

Figure 6.3.15 shows the no. of steps required by the agent to reach the goal against the no.

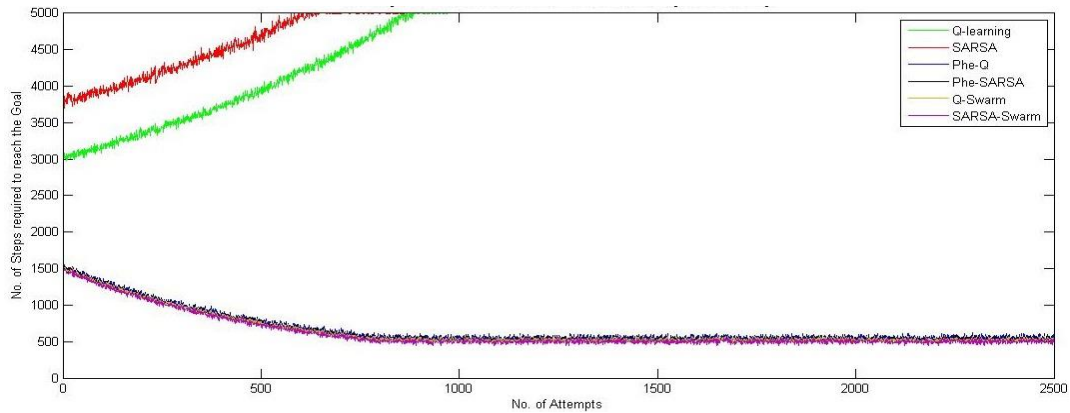


Figure 6.3.15 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case IV(a)

of attempts for a 10X10 grid world with 4 fixed and 2 moving obstacles in the path and the goal is moving in the first two rows. Six algorithms have been simulated for this case: Q-learning and SARSA Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm.

Here, for all the six methods, conventional RL methods are not able to find any optimal paths and they diverge and reach the maximum steps per attempt criterion for termination. Since, there is no coordination between the agents so they are not able to reach to a common goal. The other four methods are able to find the optimal path. But, SARSA-swarm method is able to find the shortest path for any given goal locations in the least number of attempts. This is so because, SARSA-swarm or Q-swarm, the agents coordinate with each other to tell the possible locations of the goal. The global maxima is thus here not for a particular cell, but for all the cells for which goal could possibly be present. As the number of attempts increases, the agents have an understanding of moving vertically to reach the bands where goal can be present and then traverse to look for the actual position of the goal. Phe methods also find the optimal paths for the given case. It can be seen that

mostly, the agents follow the same paths for Phe methods when near to the starting location or goal. This is so because the pheromone level is concentrated for few cells only in these regions but for other cells they a little bit distributed.

(b) No. of fixed obstacles = 5; No. of moving obstacles = 3

Figure 6.3.16 shows the no. of steps required by the agent to reach the goal against the no. of attempts for a 10X10 grid world with 5 fixed and 3 moving obstacles in the path and the goal is moving in the first two rows. Four algorithms have been simulated for this case: Q-learning and SARSA Phe-Q and Phe-SARSA.

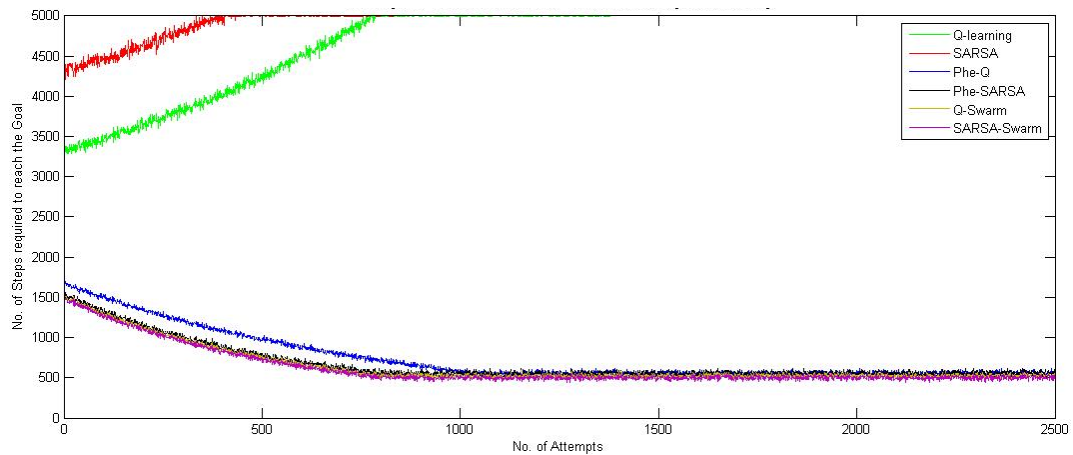


Figure 6.3.16 Plot between No. of Steps required to reach the Goal and No. of Attempts for 4 agents; case IV(b)

Here also for all the six methods, conventional RL methods are not able to find any optimal paths and they diverge and reach the maximum steps per attempt criterion for termination. Since, there is no coordination between the agents so they are not able to reach to a common goal. The other four methods are able to find the optimal path. But, SARSA-swarm method is able to find the shortest path for any given goal locations in the least number of attempts. With the increase in the number of obstacles in the path, the convergence to find the optimal path requires more number of attempts. This happens because as the number of obstacles increases, the agents are not able to explore the grid world that efficiently.

A comparative analysis for the computation time taken by all the six algorithms for the different cases simulated is provided in the following table:

Table 6.3 Computational Time for the Four Agent Problem for the six algorithms simulated

CASE	Q-Learning	SARSA	Phe-Q	Phe-SARSA	Q-Swarm	SARSA-Swarm
I	0.016846954	0.00721951	0.014432	0.014724926	0.01831005	0.016841973
II	0.307904966	0.00686411	0.015183	0.015555234	0.0181006	0.017941498
III	0.059674389	0.0068253	0.015107	0.015340303	0.01855844	0.01822305
IV	0.554626697	0.52910923	0.866702	0.859456855	1.02838567	1.02433218
V	0.674589137	0.64478662	0.913907	0.901289866	1.03895063	1.037110195
VI	Inf	Inf	0.83223	0.831296494	0.91531039	0.914505381
VII	Inf	Inf	0.9339	0.932823669	1.01634694	1.014970423
VIII	Inf	Inf	1.127067	1.126588562	1.21307443	1.213335691
IX	Inf	Inf	1.958967	1.953025083	2.02581237	2.025650189
X	Inf	Inf	2.164804	2.164227506	2.32498028	2.330970864

The computational time table shows that the SARSA computed fastest out of all the six algorithms used when the obstacles and goal both are fixed. Q-learning is slower than SARSA because for Q-learning for the Q updation formula, we use a max function which uses more memory for computation whereas in SARSA such a function is not required. The cases with dynamic obstacles or goal, conventional RL methods fail to reach the goal and hence infinite time to reach to goal. The overall time computation for the Phe-Q or Phe-SARSA is around two times that of the conventional RL, this is so because for these methods the ant needs to reach the goal and come back again to the initial starting location before the next simulation is carried out. For the swarm algorithms, as the number of parameters that needs to be calculated increases, for each attempt run the velocity parameters, global and personal best Q values are calculated and also updated. This increases the overall time of simulation for the swarm algorithms.

## CHAPTER 7

### CONCLUSIONS AND FUTURE SCOPE OF WORK

This chapter discusses the main conclusions drawn out of this thesis work and outlines the scope of future research work in the same context.

#### 7.1 MAIN CONCLUSIONS

In this thesis, study has been performed for optimal path planning for a system having one, two and four agents using Q-learning, SARSA, Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm methods. Various cases were taken where the obstacles introduced in the path were fixed and also later moving obstacles were introduced which kept on moving during the path navigation of the agent. For some cases, the goal was also made to move for a set of locations; the goal was fixed for an attempt and once the agent reaches the goal, for the next attempt the goal location was changed.

For single agent problem, when the obstacles and goal locations were fixed, all the methods stated converged to give an optimal path for 18 steps but the Phe-SARSA method gave the best convergence characteristics. In terms of computational time, SARSA method took least time for the simulation. When two and four agents were made to navigate in the same situations, SARSA-Swarm gave the best convergence characteristics. For two agents, the optimal path was found with 19 steps and for four agents, it was found to be 22 steps.

When the moving obstacles were introduced in the environment, SARSA gave the best convergence characteristics for single agent problem and also took the least computational time. The optimal number of steps to reach the goal for the single agent was about 50 for the Q-learning and SARSA methods and about 190 for the Phe-Q and Phe-SARSA methods. For the same cases, when two agents were made to navigate, SARSA-Swarm gave the best convergence characteristics. The optimal number of steps to reach the goal was about 60 for the Q-learning and SARSA, about 100 for the Q-Swarm and SARSA-Swarm and about 200 for the Phe-Q and Phe-SARSA. For the four agents problem, SARSA-Swarm gave the best convergence characteristics and the least computational time.

Q-learning, SARSA, Phe-Q and Phe-SARSA initially diverged as number of attempts increases and finally converged to a value of about 1000 steps for Phe-Q and Phe-SARSA and about 300 steps for Q-learning and SARSA. Q-Swarm and SARSA-Swarm methods also converged to a value of about 300 steps.

For the environment which had a moving goal, when one agent was made to navigate and search for goal, the optimal paths to reach the goal was 9-25 steps for all the four methods: Q-learning, SARSA, Phe-Q and Phe-SARSA. The best convergence characteristic was obtained using Phe-SARA but the computational time was obtained for SARSA. For the two agents problem, same cases were simulated to obtain optimal paths with 17-53 steps and best convergence characteristics were obtained by SARSA-swarm method. For the four agents problem, again in the same situations, optimal paths were obtained with 87-110 steps and here also best convergence characteristics were obtained by SARAS-swarm method. The conventional RL methods did not converge and failed to reach the goal as the number of attempt were increased.

When the simulation was carried out for an environment with both moving obstacles and moving goal, for single agent problem optimal path was found with about 250-500 steps for Phe-Q and Phe-SARSA and about 220-300 steps for Q-Swarm and SARSA-Swarm. For the single agent problem, best convergence characteristics and least computational time were obtained by SARSA method. For two agents problem in the same environment, it took about 400-450 steps for Q-Swarm and SARSA-Swarm methods and about 600-700 steps for the Phe-Q and Phe-SARSA methods, but best convergence characteristics were obtained using SARSA-swarm. Q-learning and SARSA methods diverged and the agents were not able to find the goal and hence diverged. For the four agent problem, in the same environment it took about 500 steps for Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm methods whereas Q-learning and SARSA methods failed to reach the goal. Best convergence results and least computational time were obtained using SARSA-swarm method.

In the cases of fixed obstacles; Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm methods always gave the better results over the Q-learning and SARSA methods. When both the obstacles and goal were kept moving, SARSA gave the best convergence

characteristics for single agent and SARSA-swarm method for multiagent problems. In those cases also, the Phe-Q and Phe-SARSA were able to find the goal. In multi-agent problem, when the goal was made moving the Q-learning and SARSA sometimes were not able to reach to goal. In these cases also, the Phe-Q, Phe-SARSA, Q-Swarm and SARSA-Swarm were able to reach the goal.

## **7.2 FUTURE SCOPE OF WORK**

The problem in this thesis was made for a fully deterministic environment, but many a times it is difficult to obtain a fully deterministic environment; for such a case the Q-value updation rule needs to take the probability of the outcome. This problem could be solved by using neural network methods and self organizing maps as this will require a mapping from the input action to the available outputs that are possible.

For the path finding problem in an unknown and dynamic environment it becomes very essential for the agent to learn the model of the system and nature of the dynamics that exist in the system. The conventional RL methods use a particular set of learning rate and the exploration rate which affects the update rule for Q-function. These parameters need to be optimized for a particular set of problem. These parameters could be optimized using bioinspired algorithms Genetic Algorithm and Fire-Fly algorithms as these methods have proven to be good optimization techniques.



## REFERENCES

- [1] M. C. Cammaerts-Tricot, J. C. Verhaeghe, “Ontogenesis of trail pheromone production and trail following behaviour in the workers of *Myrmica rubra* L. (Hymenoptera: Formicidae)”. Springer-Verlag, *Journal of Insectes Sociaux*, Vol. 21, Issue 3, pp 275-282, 1974.
- [2] D. P. Bertsekas and S. E. Shreve, “Stochastic Optimal Control: The Discrete Time Case”. Academic Press, 1978.
- [3] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels, “Self-organized Shortcuts in the Argentine Ant”. Springer-Verlag, Vol. 76, Issue 12, pp 579-581, 1989.
- [4] Christopher J.C.H. Watkins, Peter Dayan, “Technical Note: Q-Learning”, *Machine Learning Journal*, Vol. 8, Issue 3-4 , pp 279-292, 1992.
- [5] M. Dorigo, “Optimization, Learning and Natural Algorithms”, PhD thesis, Politecnico di Milano, Italy, 1992.
- [6] Michael L. Littman., “Markov games as a framework for multi-agent reinforcement learning”. *Proceedings of the Eleventh International Conference on Machine Learning*, pp 157-163, 1994.
- [7] M. L. Puterman, “Markov Decision Processes—Discrete Stochastic Dynamic Programming”. Wiley, 1994.
- [8] G. A. Rummery and M. Niranjan, “On-line Q-learning using connectionist systems”, Cambridge University Engineering Department, UK. , 1994
- [9] L. R. Leerink, S. R. Schultz, and M. A. Jabri, “A reinforcement learning exploration strategy based on ant foraging mechanisms”, *Proceedings of the Sixth Australian Conference on Neural Networks*, pp 217-220, 1995.
- [10] B. Holldobler and E. O. Wilson, “Journey of the ants: a story of scientific exploration”, Harvard University Press, 1995.

- [11] J. Kennedy, R. Eberhart, "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks IV. pp 1942–1948, 1995.
- [12] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-Dynamic Programming", Athena Scientific, 1996.
- [13] J. Kennedy, "The particle swarm: social adaptation of knowledge". Proceedings of IEEE International Conference on Evolutionary Computation. pp 303–308, 1997.
- [14] Y. Shi, R. C. Eberhart, "A modified particle swarm optimizer". Proceedings of IEEE International Conference on Evolutionary Computation. pp 69–73, 1998.
- [15] R. C. Arkin, "Behavior Based Robotics", The MIT Press, 1998.
- [16] R. S. Sutton and A. G. Barto, "Reinforcement Learning : An Introduction", Cambridge, MA: MIT Press, 1998.
- [17] Junling Hu and Michael P. Wellman, "Experimental results on q-learning for general-sum stochastic games". Proceedings of the Seventeenth International Conference on Machine Learning, ICML, San Francisco, CA, USA, pp 407-414, 2000.
- [18] S. Singh, T. Jaakkola, M. L. Littman and C. Szepesvari, "Convergence results for single-step on-policy reinforcement-learning algorithms", Journal on Machine Learning, Vol. 38, Issue 3, pp 287–308, 2000.
- [19] R. T. Vaughan, K. Stoy, G. S. Sukhatme and M. J. Mataric, "Whistling in the dark: cooperative trail following in uncertain localization space", Proceedings of the Fourth International Conference on Autonomous Agents, Barcelona, Spain, pp 187-194, 2000.
- [20] Michael L. Littman, "Value-function reinforcement learning in Markov games" Journal of Cognitive Systems Research, pp 55-66, 2001.
- [21] Michael L. Littman, "Friend-or-Foe q-learning in general-sum games", Proceedings of the Eighteenth International Conference on Machine Learning, ICML, San Francisco, pp 322-328, 2001.

- [22] H. Van Dyke Parunak and S. Brueckner, “Ant-like missionaries and cannibals: synthetic pheromones for distributed motion control”, Proceedings of the Fourth International Conference on Autonomous Agents, pp 467–474, 2001.
- [23] H. Van Dyke Parunak, S. Brueckner, J. Sauter and J. Posdamer, “Mechanisms and military applications for synthetic pheromones”, Proceedings of the Workshop on Autonomy Oriented Computation at the Fifth International Conference on Autonomous Agents, pp 58–67, 2001.
- [24] H. Van Dyke Parunak, S. Brueckner and J. Sauter, “Digital pheromone mechanisms for coordination of unmanned vehicles”, Proceedings of the First International Joint Conference on Autonomous Agents and Multi-agent Systems, ACM Press, pp 449–450, 2002.
- [25] R. T. Vaughan, K. Stoy, G. S. Sukhatme and M. J. Mataric, “LOST: localization space trails for robot teams”, IEEE Transactions on Robotics and Automation, Special Issue on Advances in Multi-robot Systems, pp 796–812, 2002.
- [26] N. Monekosso and P. Remagnino, “An analysis of the pheromone based Q-learning algorithm”, Advances in Artificial Intelligence, Proceedings of the Eight Ibero-American Conference on Artificial Intelligence, Heidelberg: Springer, pp 224-232, 2002.
- [27] Junling Hu and Michael P. Wellman, “Nash q-learning for general-sum stochastic games”, Journal of Machine Learning Research, pp 1039-1069, 2003.
- [28] N. Monekosso and P. Remagnino, “The analysis and performance evaluation of the pheromone Q-learning algorithm”, Journal of Expert Systems, Vol. 21, Issue 2, pp 80-91, 2004.
- [29] H. Lima and Y. Kuroe, “Reinforcement Learning through Interaction among Multiple Agents”, IEEE SICE-ICASE International Joint Conference, pp 2457-2462, 2006.
- [30] H. Lima and Y. Kuroe, “Swarm Reinforcement Learning Algorithms based on SARSA method”, IEEE SICE Annual Conference, pp 2045-2049, 2007.

- [31] L. Busoniu, R. Babuska, and B. D. Schutter, “A Comprehensive Survey of Multiagent Reinforcement Learning”, IEEE transactions on systems, man, and cybernetics- part c: applications and reviews, Vol. 38, no. 2, pp 156-172, March 2008.
- [32] Chia-Feng Juang, Chun-Ming Lu, “Reinforcement fuzzy control using Ant Colony Optimization”, IEEE International Conference on Systems, Man and Cybernetics, pp 927-931, 2008.
- [33] D. Kadlec and P. Nahodil, “Adopting animal concepts in hierarchical reinforcement learning and control of intelligent agents”, Proceedings of the 2nd Biennial International Conference on Biomedical Robotics and Biomechanics pp 924-929, 2008.
- [34] L. Busoniu, R. Babuska, B. D. Schutter and D. Ernst, “Reinforcement learning and dynamic programming using function approximators”, CRC Press, 2009.
- [35] Wei Wu, Geng Haifei and Jiang An, “A Multi-agent Traffic Signal Control System Using Reinforcement Learning”, Fifth International Conference on Natural Computation, 2009. ICNC '09, Vol.4, pp 553-557, 2009.
- [36] J. Papis and M. G. Lagoudakis, “Learning continuous-action control policies”, IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, pp 169-176, 2009.
- [37] Shu Da Wang, Shuo Ning Wang and Wei Ping Zhang, “Study on Multi-agent Simulation System Based on Reinforcement Learning Algorithm”, WRI World Congress on Computer Science and Information Engineering, Vol.5, pp 523-527, 2009.
- [38] Evangelos A. T., Jonas B. and Stefan S., “A Generalized Path Integral Control Approach to Reinforcement Learning”, Journal of Machine Learning Research, Vol.11, pp 3137-3181, 2010.
- [39] J. Glascher, N. Daw, P. Dayan and J. P. O'Doherty, “States versus Rewards: Dissociable Neural Prediction Error Signals Underlying Model-Based and Model-Free Reinforcement Learning”, ScienceDirect, Vol. 66, Issue 4, pp 585–595, 2010.

- [40] Q. P. Lau, Mong Li Lee and W. Hsu, "Distributed Coordination Guidance in Multi-agent Reinforcement Learning", 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI), pp 456-463, 2011.
- [41] F. T. Romero, G. R. Villanueva and I. A. Bautista, "Robotic system for reactive navigation in dynamic environments", 21st International Conference on Electrical Communications and Computers (CONIELECOMP), pp 20-205, 2011.
- [42] S. Arai and T. Miura, "An intelligent agent for combinatorial auction", 11th International Conference on Hybrid Intelligent Systems (HIS), pp 24-29, 2011.
- [43] M. Stoica, F. Sisak and A. D. Morosan, "Reinforcement learning algorithm for industrial robot programming by demonstration", 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM), pp 1517-1524, 2012.
- [44] Ji-Hwan Son and Hyo-Sung Ahn, "Bio-insect and artificial robot interaction using cooperative reinforcement learning", IEEE International Symposium on Intelligent Control (ISIC), pp 1190-1194, 2012.
- [45] D. Grady, M. Moll, L. E. Kavraki, "Automated Model Approximation for Robotic Navigation with POMDPs", IEEE International Conference on Robotics and Automation, pp 78-84, 2013.
- [46] S. Zhiguo, T. Jun, Z. Qiao, Z. Xiaomeng and W. Junming, "The Improved Q-Learning Algorithm based on Pheromone Mechanism for Swarm Robot System", IEEE 32<sup>nd</sup> Chinese Control Conference (CCC), pp 6033-6038, 2013.
- [47] Chun-Tse Lin , Hsin-Han Chiang , and Tsu-Tian Lee, "A Practical Fuzzy Controller with Q-learning Approach for the Path Tracking of a Walking-aid Robot", SICE Annual Conference, pp. 888-893, 2013.
- [48] M. I. Abouheaf and F. L. Lewis, "Multi-agent differential graphical games: Nash online adaptive learning solutions", IEEE 52nd Annual Conference on Decision and Control (CDC), pp 5803-5809, 2013.

- [49] O. Krigolson, C. Hassall and T. Handy, “How We Learn to Make Decisions: Rapid Propagation of Reinforcement Learning Prediction Errors in Humans”, *IEEE Journal of Cognitive Neuroscience*, Vol. 26 , Issue: 3 , pp 635-644, 2014.
- [50] R. Figueroa, A. Faust, P. Cruz, L. Tapia, and R. Fierro, “Reinforcement learning for balancing a flying inverted pendulum”, *11th World Congress on Intelligent Control and Automation (WCICA)*, pp 1787-1793, 2014.
- [51] J. S. Campbell, S. N. Givigi, H. M. Schwartz, “Multiple-model Q-learning for stochastic reinforcement delays”, *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp 1611-1617, 2014.
- [52] B. Zuo, J. Chen, L. Wang and Y. Wang, “A reinforcement learning based robotic navigation system”, *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp 3452-3457, 2014.
- [53] H. Tan, K. Balajee and D. Lynn, “Integration of evolutionary computing and reinforcement learning for robotic imitation learning”, *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp 407-412, 2014.
- [54] Y. Zhang, C. W. de Silva, S. Dijia and X. Youtai, “Autonomous robot navigation with self-learning for collision avoidance with randomly moving obstacles”, *9th International Conference on Computer Science & Education (ICCSE)*, pp 117-122, 2014.
- [55] D. Vasquez, B. Okal and K. O. Arras, “Inverse Reinforcement Learning algorithms and features for robot navigation in crowds: An experimental comparison”, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp 1341-1346, 2014.
- [56] B. Bischoff, D. Nguyen-Tuong, H. van Hoof, A. McHutchon, C. E. Rasmussen, A. Knoll, J. Peters and M. P. Deisenroth, “Policy search for learning robot control using sparse data”, *IEEE International Conference on Robotics and Automation (ICRA)*, pp 3882-3887, 2014.

[57] C. Yu, M. Zhang, F. Ren, G. Tan, “Multiagent Learning of Coordination in Loosely Coupled Multiagent Systems”, IEEE Transactions on Cybernetics, Vol.45, Issue:99, pp1, 2015.

[58] H. Modares, I. Ranatunga, F. L. Lewis, and D. O. Popa, “Optimized Assistive Human--Robot Interaction Using Reinforcement Learning”, IEEE Transactions on Cybernetics, Vol.45, Issue:99, pp1, 2015.

[59] Ji-Hwan Son and Hyo-Sung Ahn, “A Robot Learns How to Entice an Insect”, IEEE Journal on Intelligent Systems, Vol. 30 , Issue: 4 , pp 54-63, 2015.