REINFORCEMENT LEARNING AND EVOLUTIONARY ALGORITHMS FOR CONTAINER LOADING PROBLEM

By

SANI TIJJANI

Submitted to the Institute of Graduate Studies in Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Mevlana (Rumi) University, Turkey

2014

REINFORCEMENT LEARNING AND EVOLUTIONARY ALGORITHMS
FOR
CONTAINER LOADING PROBLEM

Submitted by **Sani Tijjani** in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering, Mevlana (Rumi) University, Turkey.

APPROVED BY:

Examining Committee Members:

Asst. Prof. Dr. Armağan ÖZKAYA . . . . . .
(Thesis Supervisor)

Asst. Prof. Dr. Mehmet HACIBEYOĞLU . . . . . .

Asst. Prof. Dr. Mustafa KAIIALI . . . . . .

Assoc. Prof. Dr. Halis Altun . . . . . .
Head, Department of Computer Engineering

Assoc. Prof. Dr. Ali Sebetci . . . . . .
Director, Institute of Graduate Studies in Science and Engineering

DATE OF APPROVAL (Day/Month/Year):

DECLARATION

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Sani TIJJANI

Signature:

DEDICATION

I wish to dedicate this work to my parents, family, father in law, mother in law, my wife, friends and entire people of Kano State.

ACKNOWLEDGEMENTS

I would like to thank my creator Almighty Allah, the most beneficent, and the most merciful, for all his guidance and giving while I was preparing, doing and finishing this master's thesis. May peace, mercy and blessing of Allah be upon his final messenger, Prophet Muhammad (S.A.W), upon his families and all his companions.

I wish to express my deepest gratitude to my supervisor Asst. Prof. Dr. Armağan ÖZKAYA, for his guidance, advice, criticism, encouragement and insight throughout that research. I also thank my lecturer Asst. Prof. Dr. Ihsan Ömür Bucak and my colleague Jawad Muhammad for their assistance that helped the success of this work.

I would also like to express my sincere gratitude to His Excellency the Executive Governor of Kano State Engineer Dr. Rabiu Musa Kwankwaso, who sponsored this program. May Allah reward him abundantly and protect him and his family from all evil.

I would also like to thank my parents, uncles, brothers, sisters, friends and the entire family for their endless love, encouragement and support throughout my life.

ABSTRACT

REINFORCEMENT LEARNING AND EVOLUTIONARY ALGORITHMS FOR CONTAINER LOADING PROBLEM

by

Sani Tijjani

M.Sc. Thesis, 2014

Thesis Supervisor: Asst. Prof. Dr. Armağan ÖZKAYA

Keywords: Container Loading Problem, Machine Learning, Reinforcement Learning, Temporal Difference, State-Action-Reward-State-Action, Genetic Algorithm, Ant Colony Optimization,

Container Loading Problem (CLP) is a space utilization problem subject to various constraints. An example of it is the placement of containers in storage so as to minimize the waste of space. Other constraints that may be imposed include a certain loading order and an even weight distribution. Although evolutionary algorithms have been extensively studied to solve this problem, Reinforcement Learning (RL) which is a means of learning optimal behaviors by interacting with the environment, has not received enough attention in this respect. This work explores the use of RL as an alternative for tackling CLP so as to minimize the waste of space while maximizing the number of containers. We have applied five different RL algorithms (Q-learning, TD(λ), Monte-Carlo, TDQ-learning and SARSA), and two types of evolutionary algorithms (Genetic Algorithm and Ant Colony Optimization) to solve this problem. Simulations have been carried out using MATLAB to compare these algorithms based on space utilization, number of containers, simulation time and speed of convergence. The simulation parameters are set so that the algorithms are allowed to fill in storage yards 100% with containers of different sizes. Results show that, in general, RL may not guarantee the best results, but can minimize the computational difficulty providing a simple way to solve this problem. Genetic Algorithm (GA) on the other hand gives best speed of convergence for small storage yards, and, unlike other approaches that may require complex computations, four RL

algorithms, namely Q-learning, TD(λ), Monte-Carlo and SARSA, give better speed of convergence than GA for larger storage yards used in our simulations. Ant Colony Optimization (ACO), while generally being worse than the others in terms of average convergence time, performs better than the ordinary Q-learning for the largest storage yard area used in our simulations and gives a result similar to GA. Growth of ACO's average convergence time seems to be slower than those of others, indicating that it has the potential to have better convergence times with increasing storage yard area sizes. In terms of the number of containers that can be packed into storage yard when the number of containers is not restricted, all RL algorithms give approximately the same result. As the storage yard size becomes larger, however, Q-learning starts performing better than all the other RL algorithms in this respect and Monte-Carlo gradually worsens even though it is the best of all for filling small storage yards. But in terms of simulation time (for RL algorithms only) TDQ performs the best for all storage yard area combinations, except for the smallest area in which it comes in second position after TD-lambda. This is because learning optimal value in TDQ can take place under any policy and learning rate effect. TD-Lambda comes second for all remaining combinations while MC and SARSA come third and fourth respectively.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SYMBOLS/ABBREVIATIONS

Symbol	Explanation
α	Learning rate
ε	Epsilon
γ	Discount-rate
λ	Lambda
τ	Pheromone
η	Heuristic information
ρ	Evaporation coefficient
i	Present arrangement
j	Next arrangement
a	Action
r	Reward
s	State
t	Time
ACO	Ant Colony Optimization
ACT	Total Area Covered by the Containers
ASY	Area of Storage Yard
EA	Evolutionary Algorithm
GA	Genetic Algorithm
MC	Monte-Carlo
NB	Number of Blue containers
NG	Number of Green containers
NR	Number of Red containers
QL	Q-learning (Deterministic)
RL	Reinforcement Learning
RMSE	Root-Mean-Squared-Error
SARSA	State-Action-Reward- State-Action
TD	Temporal Difference
TDQ	Q-learning (Off policy TD Control)

CHAPTER ONE

1. INTRODUCTION

1.1. Overview

With the rapid increase of container volume and how transport of goods using ships plays a very important role in the economic development of the world, improving the operational efficiency is one of the most important issues for container terminals [1,2,3]. From the applications point of view, container loading problem arises in practice as an optimization issue whenever containers have to be filled or loaded with boxes, so that the usage of the container is maximized [4].

The field of Machine Learning focuses on how to design systems that automatically improves with experience [5]. Many successful machine learning applications have been developed, ranging from software design to detecting fraudulent credit card transactions, to autonomous vehicles that learn to drive on public highways and even vehicles designed for fighting fire. There are also great advances in the theory and algorithms that form the foundations of machine learning field. As part of the machine learning field, Reinforcement Learning techniques deal with the problem about how an autonomous agent can learn to select proper actions through interacting with its system environment [6].

1.2. Container Loading Problem

Figure 1.1 A container [7]

A container which is a large metal box for transport of goods, shown in Figure 1.1, has played a vital role in world wide transportation. The Container Loading Problem (CLP) considers packing a set of rectangular boxes into a rectangular big box of fixed dimension [8]. We note that the big box might actually be a real container, but, according to the definition

given, it could also be the loading space of a truck, pallet or storage yard area which may be loaded up to a certain height [9]. CLP, therefore, is a space utilization problem subject to various constraints. An example of it is the placement of containers in storage yard so as to minimize the waste of space. Other constraints that may be imposed include a certain loading order and even weight distribution.

According to the typology cited in [9], one can categorize container loading problems into two:

- Those in which enough containers (large boxes) are available to accommodate all small items (small boxes). These categories are concerned with minimizing the value of the used containers.
- Such problems in which only a subset of the small items (small boxes) can be packed since the availability of containers (large boxes) is limited. These categories are concerned with maximizing the value of the packed items.

The former category includes Single-Stock-Size-Cutting-Stock Problem which is a process of packing of weakly heterogeneous set of cargo (consisting of diverse ingredients) into a minimum number of identical containers. Multiple-Stock-Size-Cutting-Stock Problem is a process of packing of weakly heterogeneous set of cargo into a weakly heterogeneous assortment of containers such that the value of the used containers is minimized. Residual-Cutting-Stock Problem is the process of packing of weakly heterogeneous set of cargo into a strongly heterogeneous assortment of containers (i.e. mostly similar containers) such that the value of the used containers is minimized. Other types that fall in this category are Single-Bin-Size-Bin-Packing Problem which is packing a strongly heterogeneous set of cargo into a minimum number of identical containers. Multiple-Bin-Size-Bin-Packing Problem is packing a strongly heterogeneous set of cargo into a weakly heterogeneous assortment of containers such that the value of the used containers is minimized. Residual-Bin-Packing Problem is packing a strongly heterogeneous set of cargo in to a strongly heterogeneous assortment of containers such that the value of the used containers is minimized. Open-Dimension Problem is packing a set of cargo into a single container with one or more variable dimension such that the container volume is minimized. All of these problems focus on minimizing the value of the used containers.

The latter category encompasses the Identical-Item-Packing Problem which is loading a single container with a maximum number of identical small items. Single-Large-Object-Placement Problem is about loading a single container with a selection from a weakly heterogeneous set

of cargo such that the value of the loaded items is maximized. Multiple-Identical-Large-Object-Placement Problem is to load a set of identical containers with a selection from a weakly heterogeneous set of cargo such that the value of the loaded items is maximized. Multiple-Heterogeneous-Object-Placement Problem is the loading of weakly or strongly heterogeneous set of containers with a selection from a weakly heterogeneous set of cargo such that the value of the loaded items is maximized. Under this category also included is Single-Knapsack Problem which is to fill a single container with a selection from a strongly heterogeneous set of cargo so to maximize the value of the loaded items. Multiple-Identical-Knapsack-Problem is defined as loading a set of identical containers with a selection from a strongly heterogeneous set of cargo such that the value of the loaded items is maximized. Multiple-Heterogeneous-Knapsack Problem is loading a set of weakly or strongly heterogeneous containers with a selection from a strongly heterogeneous set of cargo such that the value of the loaded items is maximized. All of these problems are concerned with maximizing the value of the loaded items.

The type of CLP considered in this thesis falls in the second category and the type that best describes it is Multiple-Identical-Knapsack Problem since we apply the algorithms in consideration to load a set of identical containers (i.e. the storage yard) with a selection from a strongly heterogeneous set of cargo (i.e. the containers of known sizes) such that the value of the loaded items is maximized.

Continuous increase in container volume and the need for rapid transport of goods make improving operational efficiency at the terminals critical. For instance, the process of loading outbound containers in a cargo port is of three stages: yard cranes pick up the desired containers from yard blocks and load them into the yard trailers, then yard trailers transport the containers to cranes, and lastly the cranes load the containers into the ships, as shown in Figure 1.2 [2].

It is difficult to optimize the whole container terminal operation with a single analytical model due to the complexity of the system. An important concern for the efficient operation of container loading is to find an algorithm that can optimize the way in which the containers are to be arranged, in order to speed up the process for economic requirements and ecological issues [10]. However, the efficient way to arrange containers involves new and specialized logistics process, a number of logistics plan and automated system to handle a great number of containers. Researchers have developed mathematical optimization models for different

sub-processes of the container terminal operation system. Researches have been carried out in order to find a procedure that leads to an optimal way of loading containers. Of these, some

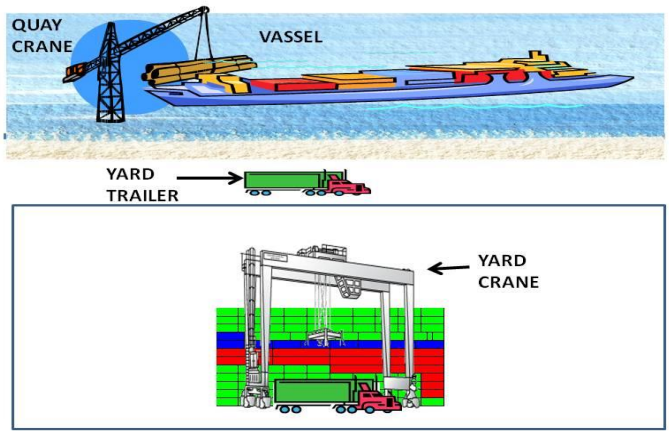


Figure 1.2 Container terminal

concern about the weight distribution, some on loading sequence or maximizing container utilization factor and others focus on optimal loading in multiple containers.

A method of integrating a Q-learning algorithm with a simulation technique is proposed to optimize the operation scheduling in the container terminals in [2]. Other related works include [11] where a simulated annealing algorithm for a single container loading problem is proposed. Aiming at the optimal layout problem of rectangular parts with dynamic constraints, a heuristic rectangular optimal layout method is proposed based on GA in [12]. An application of ACO on CLP has been also studied [8]. GA for the two-dimensional strip packing problem with rectangular pieces introduced in [13] whereas [14] suggests an approach using GA to solve the storage space allocation problem in a container terminal.

1.3 Method and Constraints

A container terminal is a facility where cargo containers are moved between different transport vehicles, for transportation [15]. At container terminals, containers are transferred from one mode of transportation to another. Within a terminal, distinct types of material handling equipment are used to transship containers from ships to barges, trucks and trains and vice versa [16].

The temporary storage of the inbound and outbound containers is one of the most important services at the container terminal that is known as the Storage Space Allocation Problem. The storage area in the terminal is divided into the several blocks of containers (Figure 1.3). Each block, as shown in Figure 1.4, consists of a number of side by side lanes with each lane including a number of containers [14].

Figure 1.3 Container terminal storage area [17]

Figure 1.4 Blocks of containers [18]

Some researchers like [13] solve CLP using two-dimensional space but in our work the width of the storage yard block is assumed to be the same with the width of all the containers, so the process space is considered as three dimensional. Hence, we have to be concerned with only heights and length of the containers. An instance of storage yard block with an arrangement of containers of different sizes is shown in Figure 1.5.

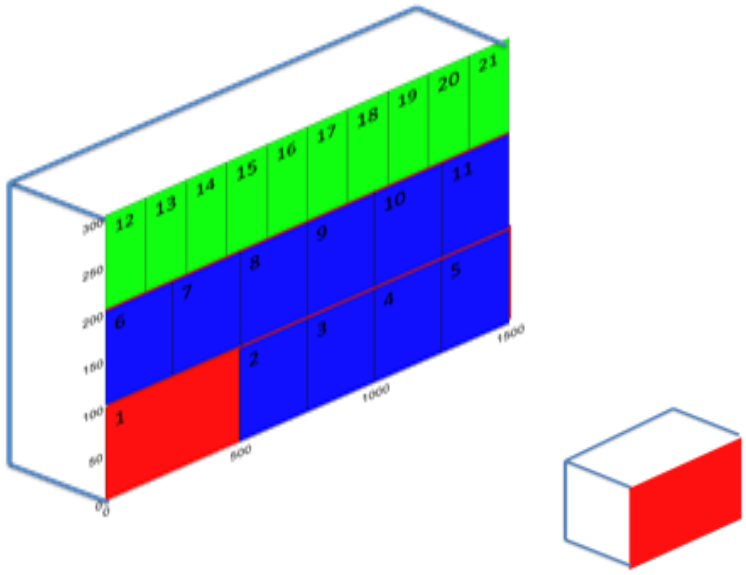


Figure 1.5 3D view of a storage yard with containers

Storage yards of dimensions of 1500x300, 1500x600, 1500x900 and 1500x12000 units are considered in this study. The sizes of containers used are 500x100, 250x100 and 150x100 which are labelled as the Red, Blue, and Green containers, respectively. Containers on ships or storage yards are stacked one on top of the other in columns, and can only be unloaded from the top of the column as assumed in [19]. Algorithms used in this work are designed to load containers layer-by-layer while holding the following conditions:

i) All containers lie entirely within the storage yard block.

ii) The containers do not overlap.

The containers are packed by stacking them one by one until the yard block is full. The first container, also called the base container, is placed at the left corner of storage yard block as shown in Figure 1.6 (a). After packing the first base container in the storage yard block, the next container is placed to fill the empty space at right side of the first container, the containers that follow are also placed at the right side of the previous container until the first row is filled as shown in Figure 1.6 (b). After the first row is filled, the next container is placed at the left corner of storage yard block on the top of the first container of the first row, as shown in Figure 1.6 (c) below, and the second row is filled similarly (Figure 1.6 (d)). The process explained above continuous until the storage yard block filled as depicted in Figure 1.6 (e) below, then we move to fill the second layer of the storage yard block.

The foundation of this work in which an approach for maximizing container loading and minimizing the waste of space using Q-learning algorithm was presented in [1]. We build our work on this paper by solving the problem using four more RL algorithms, i.e. TDQ (Off-Policy TD Control), Monte-Carlo (MC), State-Action-Reward-State-Action (SARSA), TD(λ) and also two evolutionary algorithms, i.e. Genetic Algorithm (GA) and Ant Colony Optimization (ACO) Algorithm.

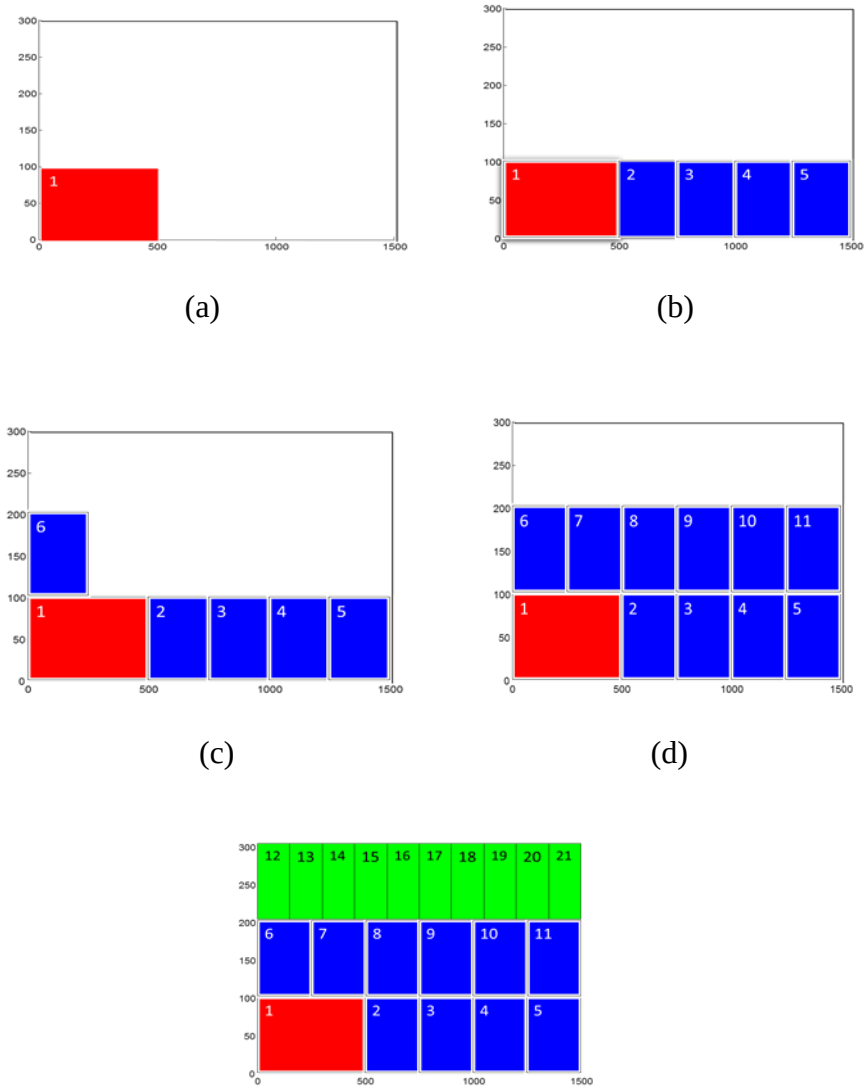


Figure 1.6 How to fill empty space in storage yard block

1.4 Organization of the Thesis

This thesis is organized as follows:

Chapter 2 describes the Reinforcement Learning approach. We discuss the Q-learning (deterministic), TDQ, Monte-Carlo, SARSA and TD(λ) algorithms as well as the procedures devised to solve the CLP.

Chapter 3 describes evolutionary algorithms. We discuss GA and ACO, and then explain in detail how these algorithms are applied to solve the CLP.

Chapter 4 presents simulations, numerical results obtained to compare the algorithms by various metrics, and discussion of results.

Chapter 5 contains conclusions and possible future work.

CHAPTER TWO

2. EVOLUTIONARY ALGORITHMS FOR CONTAINER LOADING PROBLEM

2.1. Overview

Evolutionary Algorithms (EA) is a field of artificial intelligence algorithms that attempts to bring out a system with the ability to evolve and meet new challenges or goals. EA are typically used to provide good approximate result to hard problems such as optimization techniques. Genetic Algorithm (GA) and Ant Colony Optimization (ACO) are some of the known EAs [20].

The GA is a search technique originally stimulated by biological genetics and uses ideas based on the natural genetics and biological evolution [21]. It provides an approach to learning that is based loosely on simulated evolution. Hypotheses are often described by bit strings whose interpretations depend on the application, though hypotheses may also be described by numbers, symbolic expressions or even computer programs. Genetic algorithms have been applied successfully to a variety of learning tasks and to other optimization problems [22].

ACO algorithm is a nature-inspired cybernetic method in artificial intelligence. In this technique, ants put down what is called pheromone on the ground in a form of liquid which is used for communication among them. The path that marks the presence of pheromone indicates the presence of food when followed from the nest of the ants. Many optimization problems have been solved using ACO [22,23]. ACO algorithm shows a surprisingly successful performance in the solution of NP-hard problems, which draws more and more attention to ACO research [24].

2.2. Genetic Algorithm

The search for an appropriate hypothesis begins with a population, or collection of initial hypotheses. Members of the current population give rise to the next generation population by means of operations such as random crossover and mutation, which are patterned after processes in biological evolution. At each step, the hypotheses in the current population are evaluated relative to a given measure of fitness, with the fittest hypotheses selected probabilistically as seeds for producing the next generation [5].

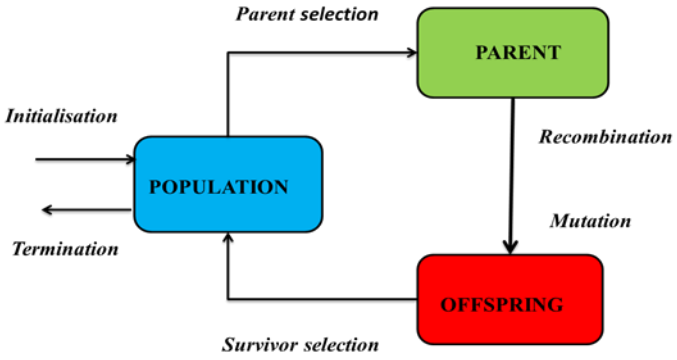


Figure 2.1 Genetic Algorithm

As can be seen from Figure 3.1 above the GA starts by generating initial population from which parent pair are also selected for recombination (cross over) and mutation. Thus, a new generation is produced as an offspring of the parent population form which the fittest hypothesis selected according to the fitness function.

2.3. GA for Container Loading

Unlike various constructive optimization algorithms that use sophisticated methods to obtain a good single solution, the GA deals with a set of solutions (population) and applies to each solution simple procedures of crossover, mutation and quality evaluation [19]. The population size used in this work is 30 (unlike RL algorithms, in GA and ACO there is a need to initialize the starting number of container arrangements. 30 is chosen because it is best suitable to produce the results that can be compared with RL algorithms). Crossover Rate is 0.6 (18 arrangements) and Mutation Rate is 0.2 (6 arrangements). Any value between 0 to 1 can be select as Crossover Rate and Mutation Rate, but in most research the Crossover Rate is mostly greater 0.5 and Mutation Rate is less than 0.5**.** The fitness function is proportional to the total number of containers placed in the storage and inversely proportional to the space wasted as calculated using the equation:

$$Fitness = \left(N_R + N_B + N_G \right) * \frac{A_{CT}}{2A_{SY}} \quad \text{..... (1)}$$

where

N_R = Number of red containers,

N_B = Number of blue containers,

N_G = Number of green containers,

A_{SY} = Area of storage yard,

A_{CT} =Total area covered by the containers

The procedure of GA for CLP is shown in Algorithm 2.1 below.

GA

Function GA
Initialize population, P in the fashion R B G
Choose Crossover Rate = 0.6 and Mutation Rate, m = 0.2
For generation = 1: maximum generation
Randomly generate container arrangement R1B1G1 to RnBnGn
While maximum fitness < fitness threshold
Select probabilistically sub population, Ps for cross over
Choose m percent of the Ps for mutation
Update P with Ps population
Compute fitness using function

$$Fitness = \left(N_R + N_B + N_G \right) * \frac{A_{CT}}{2A_{SY}}$$

End
End
Return the arrangement with the highest fitness value
End

Algorithm 2.1 Procedure of GA for CLP

2.3.1. Initial Population

30 populations were randomly generated initially as shown in Table 2.1.

Table 2.1 GA initial populations

S/N	POPULATIONS R	B	G	S/N	POPULATIONS R	B	G	S/N	POPULATIONS R	B	G
1.	1	39	23	11.	40	39	33	21.	40	39	29
2.	40	39	31	12.	40	39	39	22.	40	39	33
3.	40	39	23	13.	40	40	33	23.	38	30	38
4.	40	38	33	14.	2	37	7	24.	40	39	28
5.	39	40	30	15.	40	39	17	25.	40	12	33
6.	20	29	39	16.	40	11	33	26.	40	16	33
7.	40	38	29	17.	40	37	33	27.	40	39	33
8.	40	33	8	18.	3	39	40	28.	30	39	40
9.	40	38	30	19.	39	39	17	29.	40	39	30
10.	23	32	33	20.	40	39	40	30.	40	25	33

2.3.2. Cross Over

18 sub populations are chosen probabilistically as parent populations out of 30 initial populations for cross over (Equation 2). These are shown in bold in Table 2.1.

$$P = \frac{fitness}{\sum fitness} \quad \text{.....(2)}$$

The cross over process is carried out as follows:

39 **40** 30
20 29 **39** } ***39*** ***40*** ***39***

40 **38** 29
40 33 **8** } **40** **38** **8**

40 **38** 30
23 32 **33** } **40** **38** **33**

40 **38** 33
40 39 **39** } ***40*** ***38*** ***39***

40 **40** 33
2 37 **39** } ***40*** ***40*** ***39***

40 **39** 17
40 11 **33**
→ **40** **39** **33**

40 **37** 33
3 39 **40**
→ ***40*** ***37*** ***40***

39 **37** 17
40 39 **40**
→ ***39*** ***37*** ***40***

40 **39** 29
40 39 **33**
→ ***40*** ***39*** ***33***

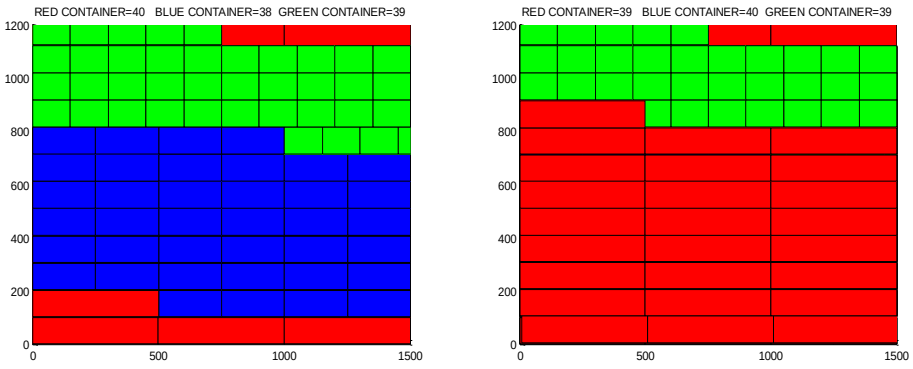
2.3.3. Mutation

Six sub populations were chosen probabilistically (Equation 2) out of 18 for cross over populations for mutation as shown below:

Table 2.2 Sub populations for mutation

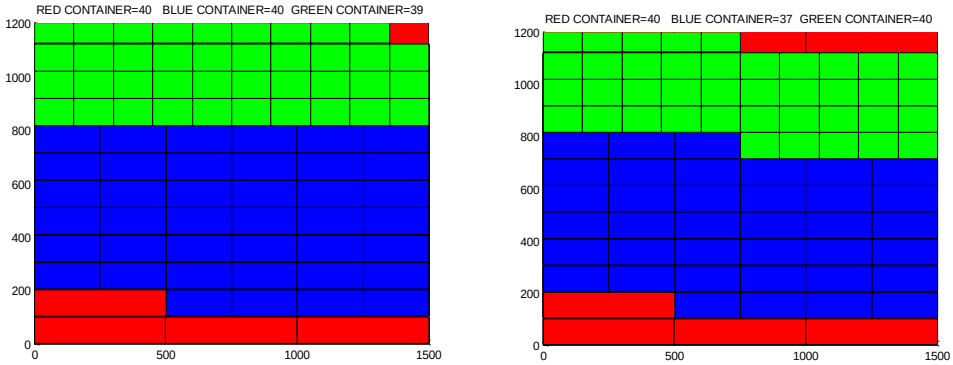
S/N	POPULATIONS		
	R	B	G
1.	39	40	39
2.	40	38	39
3.	40	40	39
4.	40	37	40
5.	39	37	40
6.	40	39	33

The swap mutation process is carried out as follows:



40 **38** **39**
39 **40** **39**
→ **39** **38** **39**

Figure 2.2 Mutation 1

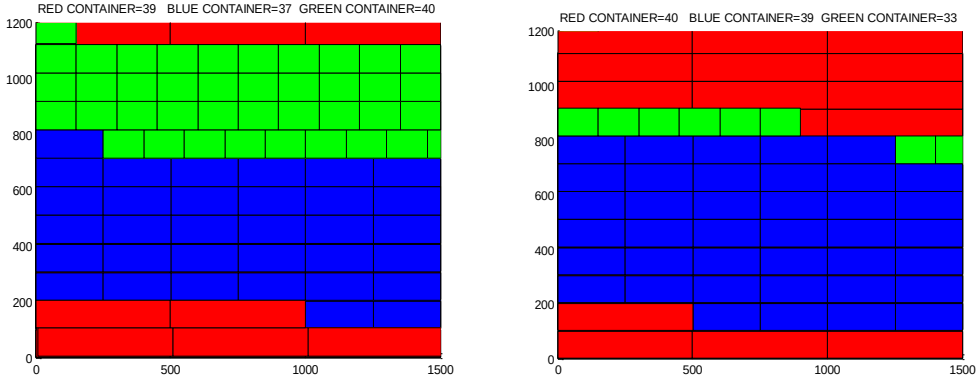


40 40 39

40 37 40

40 40 40

Figure 2.3 Mutation 2



39 37 40

40 39 33

40 39 33

Figure 2.4 Mutation 3

2.3.4. Survivor Selection

Finally the fittest arrangement is chosen out of the mutation sub populations as shown below:

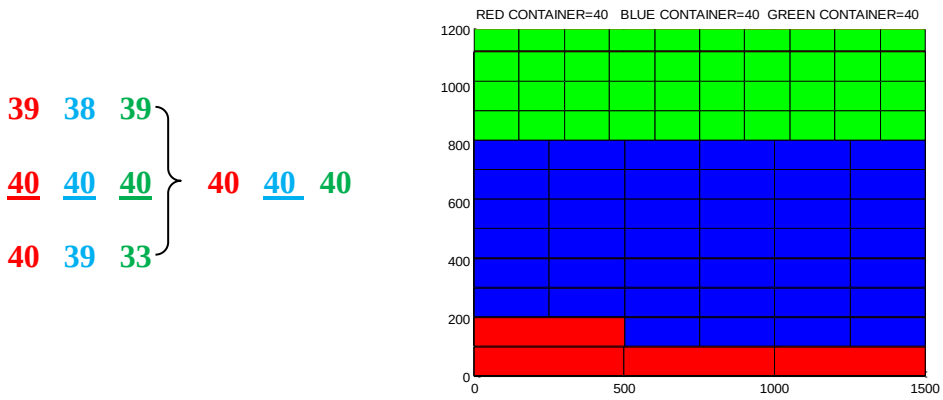


Figure 2.5 The fittest arrangement

2.4. Ant Colony Optimization (ACO) Algorithm

ACO has widely been applied to solve combinatorial optimization problems in recent years [25]. This technique is an optimization algorithm that imitates the behaviour of living ants. The ACO algorithm gives a surprisingly successful performance in the solution of NP-hard problems, which draws more and more attention on ACO research, particularly to the study of its theoretical foundation [25]. This algorithm is presented under the inspiration that an ant colony could build the shortest path from a food source to their nest by using some chemical substance called pheromone. Ants lay down pheromone trails when passing paths. The more ants choose a path, the more pheromone is laid down on it. Then, ants tend to choose the path with higher pheromone intensity. However, it is very interesting that ants do not always choose the path with the highest pheromone intensity. Otherwise, the shortest path will hardly be built up [25].

2.5. ACO for CLP

The procedure of ACO algorithm for CLP is shown in Algorithm 2.2.

After initialization step generally, in ACO algorithms two important steps are needed to be determined; the construction step and the update of the pheromone [23,26].

1. *Initialization*

   At initial stage, the artificial ACO will initial the pheromone τ . The *N* arrangements of the containers are also generated randomly. The value of *N* used in this work is 30. ACO initial populations are shown in Table 2.3.

ACO ALGORITHM

Function ACO

Initialize pheromone τ_O ***=1***

For m ants do

Randomly generate container arrangement R1B1G1 to RnBnGn

Compute heuristic information η ***using function***

$$\eta = \left(N_R + N_B + N_G\right) * \frac{ACT}{2ASY}$$

Choose next arrangements using probability function

$$P_{ij} = \frac{\tau_{ij} \times \eta_{ij}}{\sum \tau_{ij} \times \eta_{ij}}$$

Update pheromone value according to

$$\tau = \tau_{ij} + (1-\rho) \times \eta_{ij}$$

End

Return the best solution

End

Algorithm 2.2 Procedure of the ACO algorithm for CLP

Table 2.3 ACO initial populations

S/N	POPULATIONS			S/N	POPULATIONS		
	R	B	G		R	B	G
1.	1	39	23	16.	40	11	33
2.	40	39	31	17.	40	37	33
3.	40	39	23	18.	3	39	40
4.	40	38	33	19.	39	39	17
5.	39	40	30	20.	40	39	40
6.	20	29	39	21.	40	39	29
7.	40	40	39	22.	40	39	33
8.	40	33	8	23.	38	30	38
9.	40	38	30	24.	40	39	28
10.	23	32	33	25.	40	12	33
11.	40	39	33	26.	40	16	33
12.	40	39	39	27.	40	39	33
13.	40	40	33	28.	30	39	40
14.	2	37	7	29.	40	39	30
15.	40	39	17	30.	40	25	33

2. *Construction*

For 1 to maximum iteration, M ants will select one arrangement out of the *N* arrangements generated at initial stage, using the probability function:

$$P_{ij} = \frac{\tau_{ij} \times \eta_{ij}}{\sum \tau_{ij} \times \eta_{ij}} \quad \text{..... (3)}$$

where,

τ = pheromone

η = heuristic information

i = present arrangement

j = next arrangement

The heuristic information can be calculated using the equation shown below:

$$\eta = \left(N_R + N_B + N_G\right) * \frac{ACT}{2A_{SY}} \quad \text{..... (4)}$$

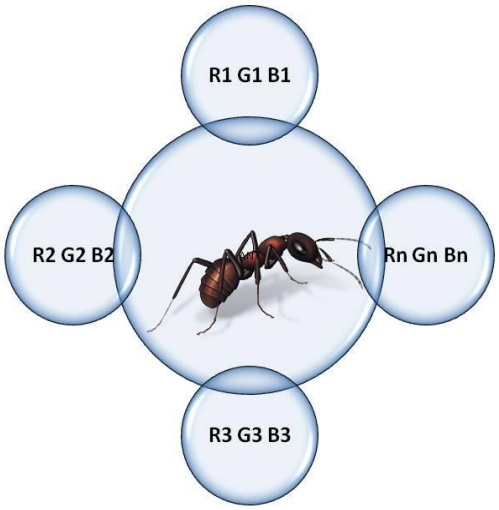


Figure 2.6 Artificial ants search for the best arrangement

The artificial ants search for best arrangement from the arrangements generated in the first generation from R1-G1-B1 to Rn-Gn-Bn. Based on the initial population given in Table 2.3, the best arrangement in the first generation is 40-40-39 which is depicted in Figure 2.7.

3. *Updating*

The pheromone value is updated at every iteration using the equation:

$$\tau_{ij} = \tau_{ij} + (1-\rho) \times \eta_{ij} \quad \text{..... (5)}$$

where

ρ =evaporation coefficient (any value between 0 to 1 can be selected).

The artificial ant chooses one of the arrangements generated above and the pheromone value is updated using Equation 5. If the arrangement chosen by the artificial ant is not the best arrangement needed, the algorithm goes to the next iteration and this process continues until the best arrangement is chosen, i.e. 40-40-40, and that is shown in Figure 2.8.

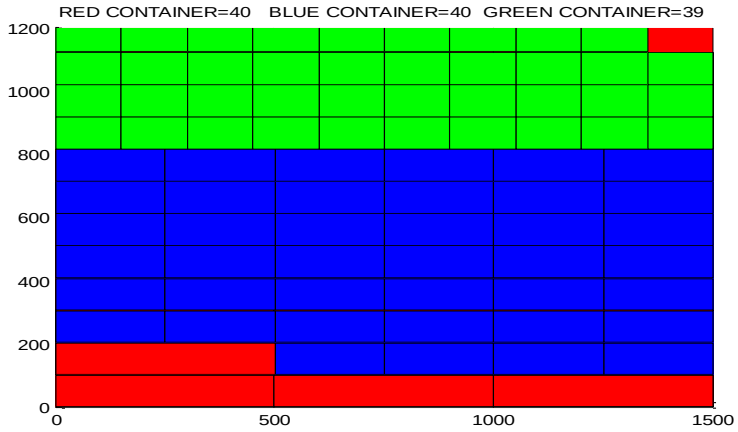


Figure 2.7 Best arrangement in the first generation

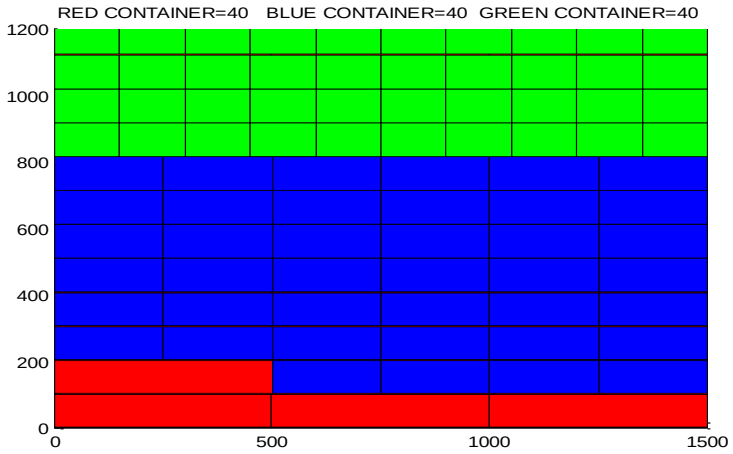


Figure 2.8 The best arrangement in all generations

CHAPTER THREE

3. REINFORCEMENT LEARNING FOR CONTAINER LOADING PROBLEM

3.1. Overview

When we think about the nature of learning, the fact that we learn by interacting with our environment is probably the first thing that comes to our mind. When an infant plays, waves its arms, or looks about, it has no explicit teacher, but it does have a direct sensor motor connection to its environment. Exercising this connection gives a lot of information about cause and effect, about the consequences of actions, and about the best actions to do in order to achieve goals. We come across such interactions of our environment throughout our lives, and we drive knowledge about what is around us. Whether a person is learning to drive a vehicle or to hold a picture or conversation, he is intensely aware of how his environment responds to what he performs, and the consequences of his actions affect his state as well as the environment he is in. Learning by interaction with an environment is the primary idea underlying nearly all theories of learning and intelligence [27].

3.2. RL Technique

RL is simply learning what to do and how to plan situations to actions, so as to maximize a return reward. In most forms of machine learning, there is no need for the learner to be told which actions to take, but instead he must discover which actions produce the highest reward by trying them [27].

Every living organism interacts with its environment and uses those interactions to improve its own actions and we call this modification of actions based on interactions with the environment Reinforcement learning (RL). Actions may influence not only the instant reward but also the next situation and, through that, all following rewards. Trial-and-error search and delayed reward are the two most important characteristics that distinguish RL from other techniques [27].

RL is a kind of unsupervised machine learning technique which belongs to the category of machine learning algorithms, and is therefore different from supervised learning methods that require a teacher [1,28]. A RL agent has the computational task of learning which action to take in a given condition (state) to achieve its goal. The learning process takes place through interaction with an environment (Fig. 3.1).

Figure 3.1 Basic Reinforcement Learning Model.

At each discrete time step, an RL agent observes the current state. In each state, the agent can take some action from the set of actions available in that state. An action can cause a transition from the current state to another state, based on the transition probabilities [1,5,28,29,30]. The model of the environment contains these transition probabilities. A numerical reward is returned back to the agent to inform it about the 'quality' of its actions or the intrinsic desirability of that state. The reward is also a part of the model of the environment.

An RL agent searches for the optimal policy in order to maximize accumulated reward [1,28,29]. An optimal policy is the sequence of actions that maximize the total reward [6]. RL deals with the problem about how an autonomous agent can learn to select proper actions through interacting with its system environment. Each time after an agent performs an action, the environment's response (as indicated by its new state) is used by the agent to reward or penalize its action [31]. The objective is to develop a decision-making policy on selecting the appropriate action rule for each agent. By RL, the optimal policy for each agent can be obtained [1,5].

In this work, we will explore RL as an alternative for tackling container loading problem, specifically for maximizing the number of containers and minimizing the waste of space. We will apply 5 known RL algorithms, namely:

- Q-learning (deterministic)
- TD(λ)
- TDQ (Off Policy TD Control)
- SARSA (On Policy TD Control)
- Monte-Carlo

3.3. Q-learning (Deterministic)

Q-learning algorithm is one of the most widely used reinforcement learning algorithms. It was proposed by Watkins in 1989 [1,2]. The objective of this algorithm is to learn the state-action pair value *Q(s, a)*, which represents the expected reward for each pair of state and action denoted by *s* and *a*, respectively. *Q(s, a)* is expressed by the following equation:

$$Q(s,a) = reward + \gamma \max_{a} Q(s', a\ ) \quad \ldots\ldots (6)$$

Before learning has started, *Q* returns a fixed value, chosen by the designer. As can be seen in Equation (6), an agent can choose an action that gives maximum reward out of all possible actions and receive reward. γ is the discount-rate parameter which impacts the present value of future rewards, in other words it determines the importance of future rewards. If we set γ=0, only the immediate or current reward is considered. As we set γ closer to 1, future rewards are given greater emphasis relative to the immediate reward. In this work, 0.9 is chosen as the value of γ.

One of the important factors of Q-learning is the action selection process. This process is responsible for selecting the actions that the agent will perform during the learning process. In this work, we consider the ε-greedy exploration. ε-greedy process selects a random action with probability ε and the best action which has the highest Q-value at the moment, with probability 1-ε [32]. The procedure of Q-learning algorithm for CLP is shown in Algorithm 3.1 below:

Q-LEARNING ALGORITHM

Function Q-learning
  Initialize Q, choose γ ***=0. 9and*** ε ***=0.1***
  For episode= 1: maximum episode
    Initialize state S for this episode
    While step <= maximum step & area covered<=goal
      Take action a ***using*** ε ***-greedy, observe reward and new state*** s'
      Update Q function according to

$$Q(s,a) = reward + \gamma \max_{a} Q(s', a\ )$$

    End
  End
  Return the best state
End

Algorithm 3.1 Procedure of the Q-learning algorithm for CLP

3.4. TD(λ)

The Q-learning learns by iteratively reducing the discrepancy between Q-value estimates for adjacent states. In this sense, Q-learning is a special case of a general class of temporal difference algorithms that learn by reducing discrepancies between estimates made by the agent at different times. TD(λ) algorithm is designed to reduce discrepancies between the estimated Q-value of a state and more distant descendants or ancestors [5]. The idea is to use a constant lambda to combine the estimates obtained from various look-ahead distances in the following fashion:

$$Q^{\lambda}(s_t, a_t) = reward + \gamma[(1-\lambda)\max_a Q'(s_t, a) + \lambda Q^{\lambda}(s_{t+1}, a_{t+1})] \quad \text{.....(7)}$$

$\max Q'(s_t, a_t)$ = is maximum future reward of Q estimates

$Q^{\lambda}(s_{t+1}, a_{t+1})$ = is look ahead value

Thus, the algorithm uses the observed reward and Q estimate to provide a perfect estimate of true Q value. If we chose λ=0, we have our original Q-learning equation which considers only one step discrepancies in the Q estimates. As λ increases, the algorithm places more emphasis on discrepancies based on more distance look-ahead. At the extreme value of λ=1, only observed "next state" reward values are considered with no contribution from the current Q estimate. The procedure of TD(λ) algorithm for CLP is shown in Algorithm 3.2 below.

TD(λ) ALGORITHM

Function TD-lambda
Initialize S, choose γ =0.9, ε =0.1 and λ =0.1
For episode= 1: maximum episode
Initialize state S for this episode
While step <= maximum step & area covered<=goal
Take action a_t using ε -greedy, observe reward and new state S_{t+1}
Update Q function according to

$$Q^{\lambda}(s_t, a_t) = reward + \gamma[(1-\lambda)\max_a Q'(s_t, a) + \lambda Q^{\lambda}(s_{t+1}, a_{t+1})]$$

End
End
Return the best state
End

Algorithm 3.2 Procedure of the TD(λ) algorithm for CLP

3.5. TDQ (Off Policy TD Control)

One of the most important breakthroughs in RL was the development of an off-policy TD control algorithm known as TDQ or Q-learning Off-Policy (Watkins, 1989). This algorithm does not pay attention to what policy (i.e. series of actions or action selection algorithm) is being followed. Instead, it just uses the best Q-value. It tries to learn the value function for the best policy, irrespective of the policy being followed. Since it uses any policy to estimate the Q-value, it is called an off-policy learning algorithm [27]. Its simplest form is defined by the equation shown below:

$$Q(s_t, a_t) = Q^{'}(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q^{'}(s_t, a_t)] \quad \text{.....(8)}$$

Here, the goal is to compute directly the optimal value function $Q(s_t, a_t)$. The core of the algorithm is a simple "value iteration update" by using the old value and making a correction based on the new information. In off-policy methods, learning optimal value can take place under any policy regarding the subject. In this case, the learned action-value function directly approximates the optimal action-value function, independent of the policy being followed [27]. The learning rate alpha determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information. The procedure of TDQ (off-policy TD control) algorithm for CLP is shown in Algorithm 3.3 below:

TDQ ALGORITHM

Function TDQ
Initialize Q, choose γ =0.9, ε =0.1 and α =0.1
For episode= 1: maximum episode
Initialize state S for this episode
While step <= maximum step & area covered<=goal
Choose a_t from S_t using ε -greedy
Observe reward and new state S_{t+1}
Update Q function according to

$$Q(s_t, a_t) = Q^{'}(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q^{'}(s_t, a_t)]$$

End
End
Return the best state
End

Algorithm 3.3 Procedure of the TDQ algorithm for CLP

3.6. SARSA (On Policy TD Control)

Unlike the previous approaches where we consider transitions from state to state and learn the values of states, we now consider transitions from state-action pair to state-action pair and learn the value of state-action pairs using the equation:

$$Q(s_t, a_t) = Q^{'}(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q^{'}(s_t, a_t)] \quad \text{.....(9)}$$

This update is done after every transition from a non terminal state s_t. Q-learning backs up the best Q-value from the state reached while SARSA waits until an action is taken and then backs up the Q-value from that action. If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1})$ is defined as zero. This rule uses every element of the quintuple of events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ that make up a transition from one state-action pair to the next [27]. This quintuple gives rise to the name "State-Action-Reward State-Action" (SARSA) for the algorithm. SARSA is an **on-policy** learning algorithm. It updates value functions strictly on the basis of the experience gained from executing some (possibly non-stationary) policy. A SARSA agent will interact with the environment and update the policy based on actions taken. It can start with a random policy, then iteratively improve and converge to optimal. On-policy learns the value function of the policy being followed [27]. The procedure of SARSA for CLP is shown in Algorithm 3.4 below:

SARSA ALGORITHM

Function SARSA
Initialize Q, Choose γ =0.8, ε =0.1 and α =0.1
For episode= 1: maximum episode
Initialize state S for this episode
Choose a from s using ε -greedy
Take action a, observed reward and new state s'
While step <= maximum step & Area covered<=Goal
Choose a' from s' using ε -greedy
Update Q function according to

$$Q(s_t, a_t) = Q^{'}(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q^{'}(s_t, a_t)]$$

End
End
Return the best state
End

Algorithm 3.4 Procedure of the SARSA algorithm for CLP

3.7. Monte-Carlo

Policy evaluation algorithms are intended to estimate the value functions for a given policy. Typically these are on-policy algorithms, and the considered policy is assumed to be stationary. Direct Monte-Carlo methods are the most straight-forward, and considered here for comparison with other methods. Monte-Carlo methods are based on simple idea of averaging a number of random samples of returns (i.e reward). Monte Carlo methods are suitable for learning from experience that does not require prior knowledge of the environment's dynamics. These methods solve the reinforcement learning problem based on averaging sample returns.

$$Q(s_t,a_t) \leftarrow Q^{'}(s_t,a_t) + \frac{1}{n(s)}\left[R_t + Q^{'}(s_t,a_t) \right] \quad \text{.....(10)}$$

$n(s)$ =number of first visit to state s.

A constant α can be formulated in order to estimate the actual return R_t as shown in the equation below:

$$Q(s_t,a_t) \leftarrow Q^{'}(s_t,a_t) + \alpha\left[R_t + Q^{'}(s_t,a_t) \right] \quad \text{.....(11)}$$

R_t = is long term reward

$\alpha\left[R_t + Q^{'}(s_t,a_t) \right]$ = updated value function

In MC approach, values for each state or pair state-action are updated only based on final reward, not on estimates of neighboring states as in previous approaches [27]. The procedure of Monte-Carlo for CLP is given in Algorithm 3.5.

3.8. Agent, Environment, State, Actions, and Reward

3.8.1. Agent

When we come to the real life application, a device that is used to load the containers can be regarded as our agent. Since in a container terminal, a yard crane is a device used to load containers in storage yard, it can be considered as our agent (Figure 3.2).

3.8.2. Environment

The problem environment is formed by the storage yard block and containers (Figure 3.3).

MONTE-CARLO ALGORITHM

Function Monte-Carlo
Initialize Q, ε ***=0.1 and*** α =0.1
For episode= 1: maximum episode
Initialize state S for this episode
While step <= maximum step & area covered<=goal
Choose a_t ***from*** s_t ***using*** ε ***-greedy***
Take action a_t ***, observe reward and new state*** s_{t+1}
Update Q function according to

$$Q(s_t, a_t) \leftarrow Q'(s_t, a_t) + \alpha\left[R_t + Q'(s_t, a_t)\right]$$

End
End
Return the best state
End

Algorithm 3.5 Procedure of the Monte-Carlo algorithm for CLP

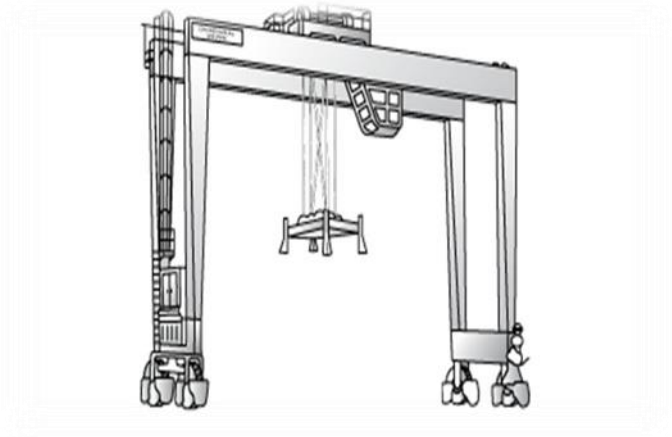


Figure 3.2 Yard crane

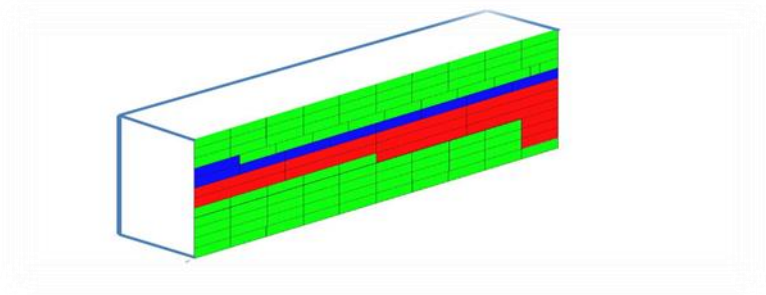


Figure 3.3 Storage yard block and containers

3.8.3. State

A state is the numbers of red containers, blue containers, and green containers together with their positions in the storage yard at a given time. Thus, states are the ways the containers arranged in the storage yard and the number of containers contained in the storage yard block.

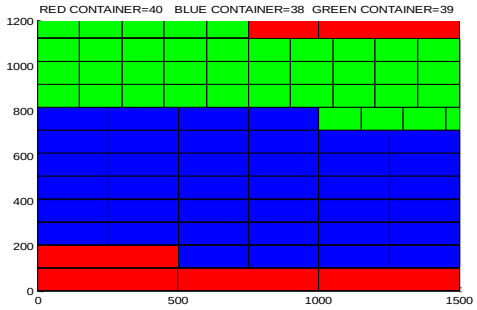


Figure 3.4 A state

3.8.4. Actions

The agent can choose any one of the possible actions shown below. For example, if removing one red container can give space to replace it with two green containers, removing that red container is a good action at this situation. Similarly if the space we have can be best filled with a single red container, so adding a red container is the best action at this particular moment. Any action applied yields to a new state.

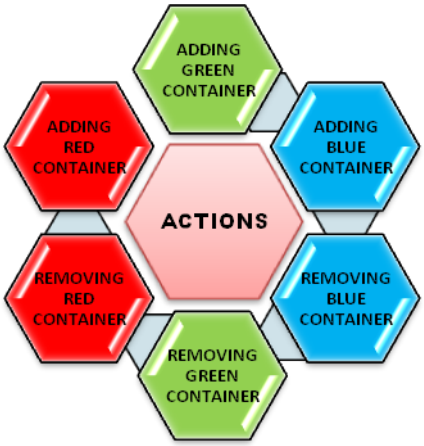


Figure 3.5 Actions

3.8.5. Reward

As we aim to minimize the waste of space (i.e. empty space) and maximize the number of containers, the reward is set to be proportional to the total number of containers placed in the storage and inversely proportional to the space wasted using the equation:

$$Reward = \left(N_R + N_B + N_G\right) * \frac{A_{CT}}{2A_{SY}} \quad \text{..... (12)}$$

where

N_R = Number of red containers,

N_B = Number of blue containers,

N_G = Number of green containers,

A_{SY} = Area of storage yard,

A_{CT} =Total area covered by the containers

Note that the Area of Storage Yard (ASY) is multiplied by two in Equation (12) above (i.e. 2*ASY). This is because our algorithms are designed to fill two layers of storage yard block.

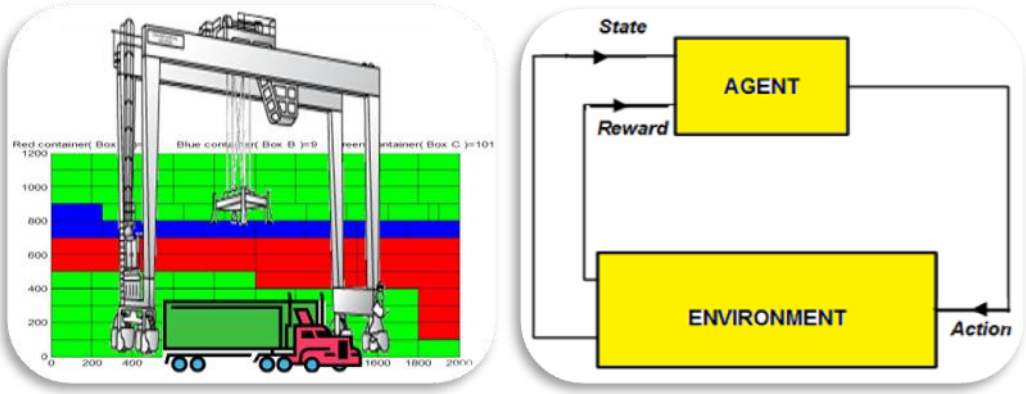


Figure 3.6 RL for container loading

In a nutshell, when an agent (yard crane) interacts with an environment (storage yard block) by applying one of the six possible actions (add R, remove R, add B, remove B, add G or remove G), it enters a new state (a new container arrangement and a new number of containers) and receives the reward as calculated in Equation (12).

CHAPTER FOUR

4. NUMERICAL EXPERIMENTS

4.1. Overview

The simulations of this work have been carried out using MATLAB software on a computer with Intel Core-i3-3110M dual core 2.4 GHz CPU and 4GB RAM running Windows8 64bit operating system. For experiments, the area of the storage yard has been chosen to be 1500x300, 1500x600, 1500x900 and 1500x1200 unit squared (usually the unit of centimeter may be used, but in practice any appropriate unit can be used regarding the area of storage yard and area of containers).

The measure used to decide the convergence of RL algorithms is the Root-Mean-Squared-Error (RMSE) which is the difference between the Q-value at an episode and the previous episode's Q-value. The number of episodes run is 500. During an episode, if a chosen action happens to be bad, it will cause an instant rise in RMSE. At convergence, RMSE is approximately zero which means the true value function is attained. The Goal of our agent is to cover 100 percent of the storage area within 500 episodes.

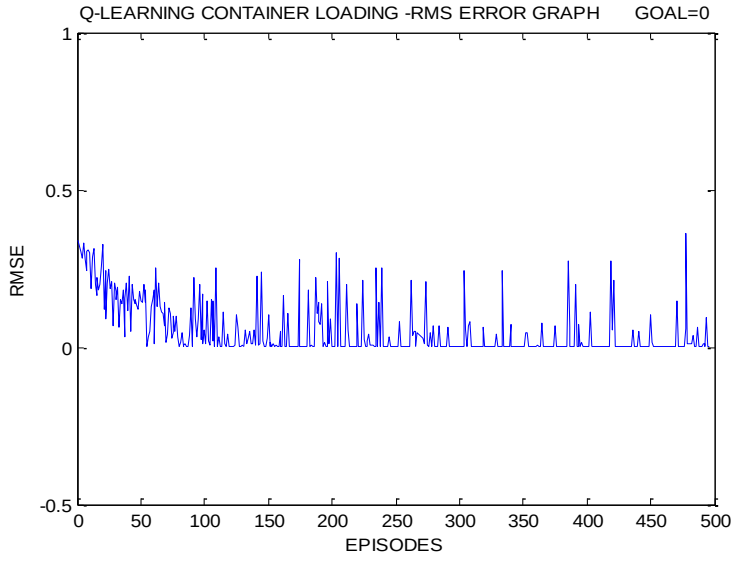


Figure 4.1 Series of bad action selections.

As can be seen from the example given in Figure 4.1, an agent makes bad action selections throughout 500 episodes and the error keeps on rising. The number of goals reached is zero since the agent fails to reach the goal state within all 500 episodes.

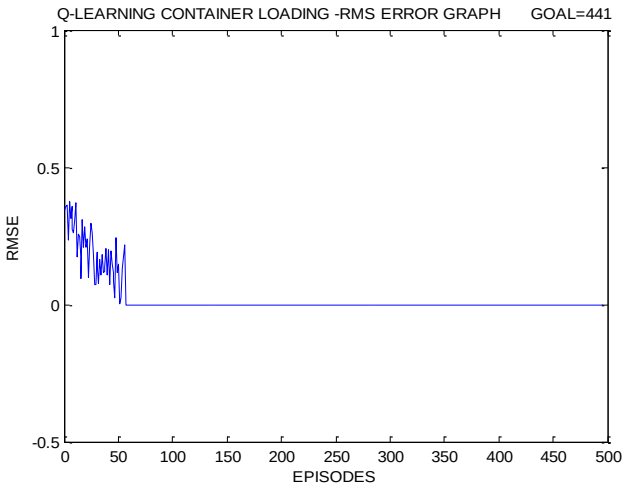


Figure 4.2 Good action selections and convergence.

On the other hand, Figure 4.2 shows another example where an agent makes good action selections throughout remaining episodes after 59 episodes. The error converges to zero due to the good action selections made by the agent. The number of goals reached is 441 since the agent fails to reach the goal state for the first 59 episodes out of 500.

A sample of a storage yard of size 1500x300 with an arrangement of containers placed to fill in the yard after running a simulation is shown in Figure 4.3.

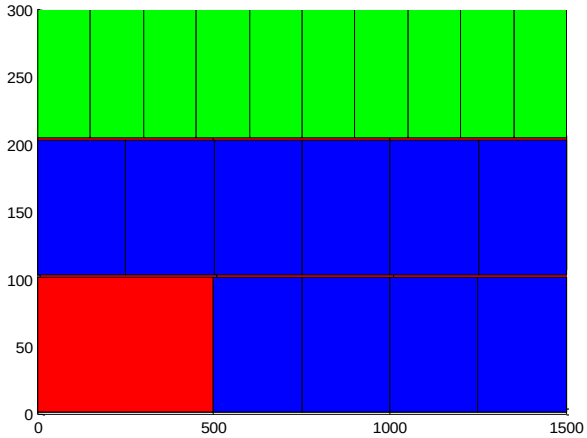


Figure 4.3 1500x300 storage yard area with an arrangement of containers

4.2. Comparison between RL Algorithms

Under this subsection the five RL algorithms are compared without restriction to the number of containers while the area of storage yard is fixed to 1500x300, 1500x600, 1500x900, and 1500x1200. The goal of our agent is to cover 100 percent of the storage area in 12 steps within 500 episodes. The size of red container is 500x100 unit squared, blue container 250x100 unit squared, and green container 150x100 unit squared.

The simulation time, the episode at which the algorithm converges, convergence period, the number of times an agent reaches the Goal state and the numbers of containers loaded were recorded for multiple runs of simulations. These are shown in the summary Tables 4.1 to 4.4 below and Appendices A to D.

Table 4.1 RL algorithms comparison summary without restriction to the number of containers (1500x300).

Algorithm	**Average convergence time (s)**	**Most frequent convergence time(s)**	**Average number of containers**
QL	0.406	0.260	31.4
TDQ	0.089	0.049	32.4
MC	0.096	0.045	32.7
SARSA	0.106	0.120	32.4
TD(λ)	0.064	0.045	32.5

Table 4.2 RL algorithms comparison summary without restriction to the number of containers (1500x600).

Algorithm	**Average convergence time (s)**	**Most frequent time (s)**	**Average number of containers**
QL	2.270	2.285	62.767
TDQ	0.692	0.589	62.6
MC	0.835	0.954	63.45
SARSA	1.353	0.896	63.1
TD(λ)	**0.519**	**0.34**	62.2

Table 4.3 RL algorithms comparison summary without restriction to the number of containers (1500x900).

Algorithm	**Average convergence time (s)**	**Most frequent time (s)**	**Average number of containers**
QL	10.569	10.593	93.5
TDQ	2.320	1.005	93.6
MC	2.648	3.440	94
SARSA	2.816	1.029	93.933
TD(λ)	2.184	0.980	94.05

Table 4.4 RL algorithms comparison summary without restriction to the number of containers (1500x1200).

Algorithm	Average convergence time (s)	Most frequent time (s)	Average number of containers
QL	33.333	25.127	126
TDQ	5.569	6.650	125.5
MC	6.083	2.315	125.7
SARSA	7.959	5.957	125.7
TD(λ)	4.930	6.411	125.3

4.3. Comparison between RL and EAs

In this section, five RL and two EA Algorithms were compared when the number of containers restricted to 30, 60, 90, and 120 while the area of storage yard is fixed to 1500x300, 1500x600, 1500x900, and 1500x1200, respectively. The goal is to cover 100 percent of the storage area within 500 episodes. The sizes of red, blue, and green containers are 500x100 unit squared, 250x100 unit squared, and 150x100 unit squared, respectively. The simulation time, the episode at which the algorithm converge, convergence period, the number an agent reaches the goal state and the number of containers loaded were recorded for multiple runs of simulations . These are shown in the summary Tables 4.5 to 4.8 below and Appendices E to H.

Table 4.5 RL and EAs comparison summary with number of containers restricted to 30 (1500x300).

Algorithm	Average convergence time (s)	Most frequent time (s)
QL	0.594	0.882
TDQ	0.110	0.050
MC	0.105	0.103
SARSA	0.109	0.089
TD(λ)	0.074	0.020
GA	0.041	0.040
ACO	0.676	0.639

Table 4.6 RL and EAs comparison summary with number of containers restricted to 60 (1500x600).

Algorithm	Average convergence time (s)	Most frequent time (s)
QL	1.423	1.478
TDQ	0.401	**0.050**
MC	0.427	0.096
SARSA	0.483	0.450
TD(λ)	0.361	0.326
GA	**0.193**	0.140
ACO	2.536	0.680

Table 4.7 RL and EAs comparison summary with numbers of containers restricted to 90 (1500x900).

Algorithm	Average convergence time (s)	Most frequent time (s)
QL	2.470	0.642
TDQ	0.759	0.667
MC	1.113	1.328
SARSA	1.202	0.913
TD(λ)	**0.632**	**0.223**
GA	1.099	0.427
ACO	4.865	6.313

Table 4.8 RL and EAs comparison summary with number of containers restricted to 120 (1500x1200).

Algorithm	Average convergence time (s)	Most frequent time (s)
QL	7.192	5.362
TDQ	1.741	2.573
MC	2.279	3.189
SARSA	2.811	2.573
TD(λ)	**1.663**	1.922
GA	4.119	**0.381**
ACO	5.178	6.731

4.4. Discussion on the Results of Comparisons between RL and EAs

For storage yard areas of 1500x300, 1500x600, 1500x900 and 1500x1200, comparisons of the performance of the RL algorithms without restriction to the number of containers and comparisons between RL and EAs by restricting number of containers to 30, 60, 90 and 120 are summarized in the following tables.

Table 4.9 RL and EAs comparison summary both with restriction and without restriction to the number of containers (1500x300).

	WITHOUT RESTRICTION			WITH RESTRICTION		
Algorithm	**Average convergence time (s)**	**Most frequent time (s)**	**Average number of containers**	**Average convergence time (s)**	**Most frequent time (s)**	**Number of containers**
QL	0.406	0.260	31.4	0.594	0.882	30
TDQ	0.089	0.049	32.4	0.110	0.050	30
MC	0.096	0.045	32.7	0.105	0.103	30
SARSA	0.106	0.120	32.4	0.109	0.089	30
TD(λ)	**0.064**	**0.045**	32.5	0.074	**0.020**	30
GA	-	-	-	**0.041**	0.040	30
ACO	-	-	-	0.676	0.639	30

Table 4.10 RL and EAs comparison summary both with restriction and without restriction to the number of containers (1500x600).

	WITHOUT RESTRICTION			WITH RESTRICTION		
Algorithm	**Average convergence time (s)**	**Most frequent time (s)**	**Average number of containers**	**Average convergence time (s)**	**Most frequent time (s)**	**Number of containers**
QL	2.270	2.285	62.767	1.423	1.478	60
TDQ	0.692	0.589	62.6	0.401	**0.050**	60
MC	0.835	0.954	63.45	0.427	0.096	60
SARSA	1.353	0.896	63.1	0.483	0.450	60
TD(λ)	**0.519**	**0.34**	62.2	0.361	0.326	60
GA	-	-	-	**0.193**	0.140	60
ACO	-	-	-	2.536	0.680	60

Table 4.11 RL and EAs comparison summary both with restriction and without restriction to the number of containers (1500x900).

	WITHOUT RESTRICTION			WITH RESTRICTION		
Algorithm	**Average convergence time (s)**	**Most frequent time (s)**	**Average number of containers**	**Average convergence time (s)**	**Most frequent time (s)**	**Number of containers**
QL	10.569	10.593	93.5	2.470	0.642	90
TDQ	2.320	1.005	93.6	0.759	0.667	90
MC	2.648	3.440	94	1.113	1.328	90
SARSA	2.816	1.029	93.93333	1.202	0.913	90
TD(λ)	**2.184**	**0.980**	94.05	**0.632**	**0.223**	90
GA	-	-	-	1.099	0.427	90
ACO	-	-	-	4.865	6.313	90

Table 4.12 RL and EAs comparison summary both with restriction and without restriction to the number of containers (1500x1200).

	WITHOUT RESTRICTION			WITH RESTRICTION		
Algorithm	**Average convergence time (s)**	**Most frequent time (s)**	**Average number of containers**	**Average convergence time (s)**	**Most frequent time (s)**	**Number of containers**
QL	33.333	25.127	126	7.192	5.362	120
TDQ	5.569	6.650	125.5	1.741	2.573	120
MC	6.083	**2.315**	125.7	2.279	3.189	120
SARSA	7.959	5.957	125.7	2.811	2.573	120
TD(λ)	**4.930**	6.411	125.3	**1.663**	1.922	120
GA	-	-	-	4.119	**0.381**	120
ACO	-	-	-	5.178	6.731	120

Table 4.13 RL algorithms average simulation time comparison without restriction to the number of containers.

Algorithm	**Storage Yard Area**			
	1500X300	**1500X600**	**1500X900**	**1500X1200**
QL	12.302	60.756	197.499	495.456
TDQ	2.235	**12.996**	**40.090**	**96.277**
MC	2.398	15.770	48.059	109.097
SARSA	2.635	17.116	46.767	127.262
TD(λ)	**2.208**	15.526	44.278	103.094

As can be observed from the results obtained using different combinations of storage yard areas, when the number of containers is not restricted all the algorithms load more containers compared to the simulations when the number of containers is restricted to 30, 60, 90 and 120 for the same storage yard. Also, the simulations take less time when the number of containers is restricted than that when the number of containers is not restricted.

As to the convergence time of the RL algorithms without restriction to the number of containers, TD(λ) leads in all the cases because this algorithm is designed to reduce discrepancies between the estimated Q-values of a state and more distant descendants or ancestors of the state. Q-learning always comes last because all the remaining algorithms are the improvement of this algorithm. For example, if we choose lambda=0 in TD(λ), we end up with the original Q-learning equation which considers only one step discrepancies in the Q estimates. TDQ and Monte-Carlo come second and third respectively, because these approaches use a learning rate which determines to what extent the newly acquired

information will override the old information. SARSA comes fourth; this algorithm also uses a learning rate but selects actions twice for each step which is why it performs better than the ordinary Q-learning algorithm.

In terms of the number of containers that can be packed into storage yard when the number of containers is not restricted, all RL algorithms give approximately the same result. As the storage yard size becomes larger, however, Q-learning starts performing better than all the other RL algorithms in this respect and Monte-Carlo gradually worsens even though it is the best of all for filling small storage yards.

In terms of the average simulation/completion time of RL algorithms, TDQ performs the best for all storage yard area combinations, except for the smallest area in which it comes in second position after TD-lambda. This is because learning optimal value in TDQ can take place under any policy and learning rate effect. TD-Lambda comes second for all remaining combinations while MC and SARSA come third and fourth respectively.

For the comparison of the performance of the GA and RL algorithms with restriction to the number of containers, GA performs best for the first two combinations of storage yard areas (1500x300 and 1500x600), followed by TD(λ), TDQ, Monte-Carlo, SARSA, ACO and finally Q-learning. As the size of storage yards increase to 1500x900 and 1500x1200, however, TD(λ), TDQ and Monte-Carlo come first, second and third respectively, leaving GA, SARSA , Q-learning and ACO in subsequent positions. The good performance of the GA for small storage yards is due to the cross-over and mutation that help the algorithm to produce best arrangement in short time. TD-Lambda follows GA for small storage yards and performs the best for large yards due to the effect of lambda which reduces discrepancies between the estimate Q-values between present and next Q-values.

CHAPTER FIVE

5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

This work explores the reinforcement learning to tackle container loading problem for maximizing the number of containers and minimizing the waste of space. We have compared five different types of such algorithms between themselves as well as against two evolutionary algorithms based on space utilization, number of containers, simulation time and speed of convergence. If the yard crane is considered as our agent, it can select the optimal loading strategy based on the real-time state of the operational system. Reducing the time at which the containers are unloaded from a ship and placed in the storage yard reduces the waiting time of the ship. Although all of the developed procedures may serve to reduce the waiting time of the ships at the container terminal, TD(λ) performs the best overall. In real-life applications, TD(λ) can be considered the most appropriate due to its high speed of convergence and minimum completion time.

Thus, we achieved the aim of this work which was to design software for container loading with aforementioned constraints using five different RL algorithms and two evolutionary algorithms to provide a comparison so to reduce the waiting time of the ship at a container terminal. We have also empirically determined which of these algorithms are advantageous in certain cases.

5.2. Future Work

The work in the future can include implementing the Bee Colony Algorithm to solve the CLP and compare its performance to those given in this thesis. We can also try to increase the complexity of the work to make it resemble more of real-life cases by adding additional types of containers, more layers, and larger storage yards. This work can be expanded to include applications and comparisons of algorithms to the other types of container loading problem specified in [9].

REFERENCES

[1] S. Tijjani and I.O. Bucak. "An approach for maximizing container loading and minimize the waste of space using Q-learning," in *Proc. International Conference on Technological Advance in Electrical, Electronics and Computer Engineering,* 2013, pp. 236-239.

[2] Q. Zeng and Z. Yang, "An Approach Integrating Simulation and Q-learning Algorithm for Operation Scheduling in Container Terminals." *Journal of the Eastern Asia Society for Transportation Studies*, vol. 8, pp. 1-15, 2010.

[3] L.J. Pires and P.R. Pinheiro "A Hybrid Methodology Approach for Container Loading Problem Using Genetic Algorithm to Maximize the Weight Distribution of Cargo" *Real world application of genetic algorithm,* University of Fortaleza (UNIFOR,) Brazil, pp.183-189, 2012.

[4] Parreño, Francisco, et al. "A maximal-space algorithm for the container loading problem." *INFORMS Journal on Computing,* Vol. 20. No. 3, pp.412-422, 2008.

[5] T.M. Mitchel, March 1997. "Machine Learning" McGraw-Hill Engineering. Pp.367-388.

[6] Alpaydin, Ethem. "Introduction to machine learning". MIT press, pp 1-395, 2004.

[7] Standard purpose container, http://www.freeline.bg/useful-information/container-types/?lang=en (Access on 30th may, 2014 by 12.46am).

[8] Yap, Ching Nei, Lai Soon Lee, Z. A. Majid, and Hsin Vonn Seow. "Ant Colony Optimization for Container Loading Problem." *Journal of Mathematics & Statistics,* Vol. 8, pp. 169-175, 2012.

[9] Bortfeldt, Andreas, and Gerhard Wäscher. "Container loading problems: A state-of-the-art review". University Magdeburg German, Faculty of Economics and Management, pp. 1-39, 2012.

[10] Dereli, Türkay, and Gülesin Sena Das. "A hybrid 'bee (s) algorithm'for solving container loading problems." *Journal of Applied Soft Computing*, Vol. 11, No. 2, pp. 2854-2862, 2011.

[11] W. Hongtao, W. Zhoujing and L. Jian "A simulated annealing algorithm for single container loading problem" *International conference on service system and service management,* pp. 551-556, IEEE 2012.

[12] M. Li "An Improved Genetic Algorithm for the Packing of Rectangles" *Eighth International Conference on Machine Learning and Cybernetics*, Baoding, pp. 1690-1692, IEEE 2009.

[13] Bortfeldt, Andreas, "A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces." *European Journal of Operational Research*, Vol. 172, No. 3, pp. 814-837, 2006.

[14] Bazzazi, Mohammad, Nima Safaei, and Nikbakhsh Javadian, "A genetic algorithm to solve the storage space allocation problem in a container terminal." *Journal of Computers & Industrial Engineering* Vol. 56, No. 1, pp. 44-52, 2009.

[15] Container terminal, http://en.wikipedia.org/wiki/Container_terminal (Access on 30th may, 2014 by 12.59am).

[16] Vis, I. F., & De Koster, R. "Transshipment of containers at a container terminal" An overview. *European Journal of operational research*, Vol. 147, No. 1, pp 1-16, 2003.

[17] Yantian Container Terminal - Phase 1 & II, China, http://www.aecom.com/Where+We+Are/Asia/Geotechnical/_carousel/Yantian+Container+Terminal+-+Phase+1+&+II,+China (Access on 30th may, 2014 by 1.02am).

[18] Z-FRACHT: Your own World of Transports**,** Used shipping Containers & Brand New Dry Containers for sale, http://zfracht.ucoz.com/news/used_shipping_containers_brand_new_dry_containers_for_sale/2013-03-21-7 (Access on 30th may, 2014 by 1.02am).

[19] Dubrovsky, Opher, Gregory Levitin, and Michal Penn. "A genetic algorithm with a compact solution encoding for the container ship stowage problem." *Journal of Heuristics* Vol. 8, No. 6, pp. 585-599, 2002.

[20] Smilkstein, Tina, et al. "An evolutionary algorithm testbed for quick implementation of algorithms in hardware." *Evolving and Self-Developing Intelligent Systems, ESDIS'09*, IEEE, pp. 51-57, 2009.

[21] Morris, Garrett M., David S. Goodsell, Robert S. Halliday, Ruth Huey, William E. Hart, Richard K. Belew, and Arthur J. Olson. "Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function." *Journal of computational chemistry*, Vol. 19, No. 14, pp. 1639-1662, 1998.

[22] R. Rajeswari and R. Rajesh, "A modified ant colony optimazation based approach for image edge detection" *International Conference on Image Information Processing*, pp. 1-6, 2011.

[23] Muhammad, A., Bala, I., Salman, M. S., & Eleyan, A. "Discrete wavelet transform-based Ant Colony Optimization for edge detection". *International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, pp. 280-283,), May, 2013.

[24] Randall, M., & Tonkes, E. "Solving network synthesis problems using ant colony optimisation. Engineering of Intelligent Systems", Springer Berlin Heidelberg,, pp. 1-10, 2001.

[25] Huang, H., Wu, C. G., & Hao, Z. F "A pheromone-rate-based analysis on the convergence time of ACO algorithm" *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 39, pp. 910-923, 2009.

[26] T. ling, Y. Weiyu, C. Li and M. Lihong, "Image edge detection using variation-adaptive ant colony optimization," *Transactions on Computational Collective Intelligence V Lecture Notes in Computer Science*, Vol. 69, No. 10, pp. 27-40, 2011

[27] RS. Sutton and AG. Barto, 1998. " Reinforcement Learning, an introduction" A Bradford Book, The MIT Press, Cambridge, Massachusetts London, England, pp. 9-397.

[28] I.O. Bucak, and M.A. Zohdy, 2001. "Reinforcement learning control of nonlinear multi-link system", *Engineering Applications of Artificial Intelligence*, *Journal of IFAC*, Vol. 14, pp. 563-575.

[29] T. Wauters, November, 2012. "Reinforcement learning enhanced heuristic search for combinatorial optimization" Katholieke Universiteit Leuven, Faculty of Engineering Celestijnenlaan Heverlee, Belgium, pp. 7-14.

[30] I.O. Bucak and M.A. Zohdy, "Application of reinforcement learning control to a nonlinear dextereous robot", *The 38th IEEE Conference on Decision and Control (CDC'99)*, Phoenix, Arizona, Dec. pp. 5108-5113, 1999.

[31] F.L. Lewis and D. Vrabie, "Reinforcement Learning and Adaptive Dynamic Programming for Feedback Control " *IEEE Circuit andsystems magazine*, pp. 40-58, 2009.

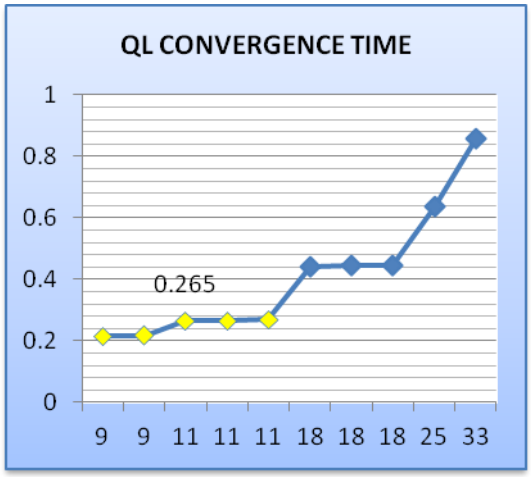
[32] Rodrigues Gomes, Eduardo, and Ryszard Kowalczyk. "Dynamic analysis of multiagent Q-learning with ε-greedy exploration." *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, PP. 369-376, 2009.

APPENDICES

APPENDIX A: Comparison between RL algorithms without restriction to the number of containers (1500x300).

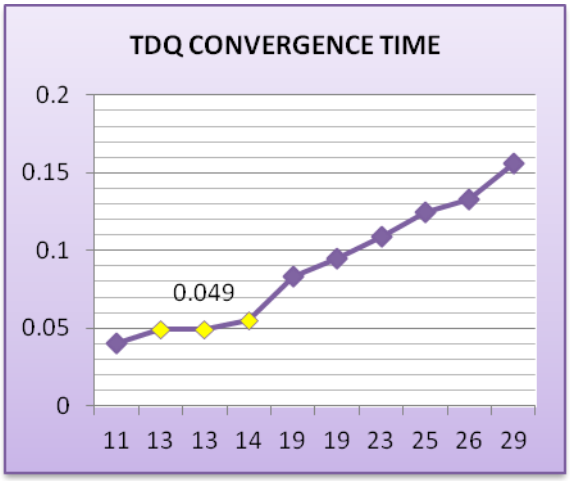
Appendix A1: Q-learning

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	12.219	11	0.269	490	31
2.	13.012	33	0.859	468	31
3.	12.032	11	0.265	490	31
4.	12.026	11	0.265	490	33
5.	12.726	25	0.636	476	32
6.	12.319	18	0.443	483	29
7.	12.334	18	0.444	483	34
8.	12.028	9	0.217	492	29
9.	12.400	18	0.446	483	32
10.	11.926	9	0.215	492	32



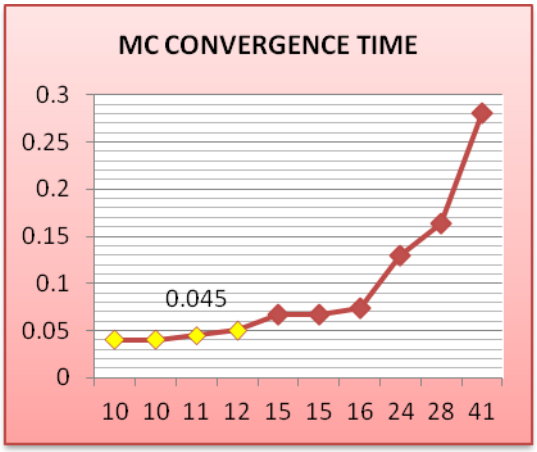
Appendix A2: TDQ

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	2.555	26	0.133	475	31
2.	2.187	19	0.083	482	32
3.	1.947	14	0.055	487	33
4.	1.897	13	0.049	488	33
5.	1.899	13	0.049	488	33
6.	1.802	11	0.040	490	34
7.	2.697	29	0.156	472	33
8.	2.506	19	0.095	482	33
9.	2.364	23	0.109	478	32
10.	2.495	25	0.125	476	30



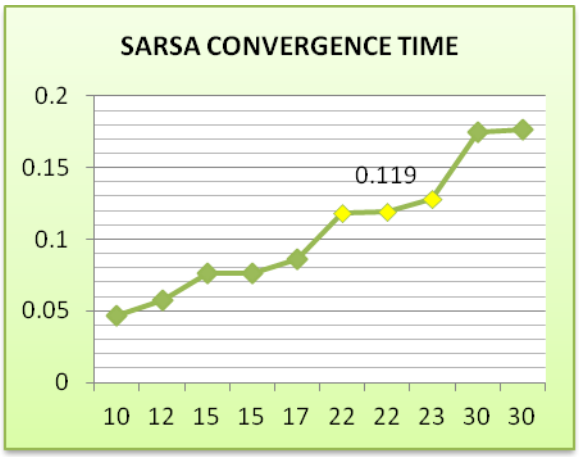
Appendix A3: Monte-Carlo

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	1.998	10	0.040	491	32
2.	3.425	41	0.281	460	33
3.	2.694	24	0.129	477	32
4.	1.982	10	0.040	491	33
5.	2.081	12	0.050	489	31
6.	2.054	11	0.045	490	33
7.	2.916	28	0.163	473	33
8.	2.317	16	0.074	485	31
9.	2.229	15	0.067	486	35
10.	2.286	15	0.067	486	34



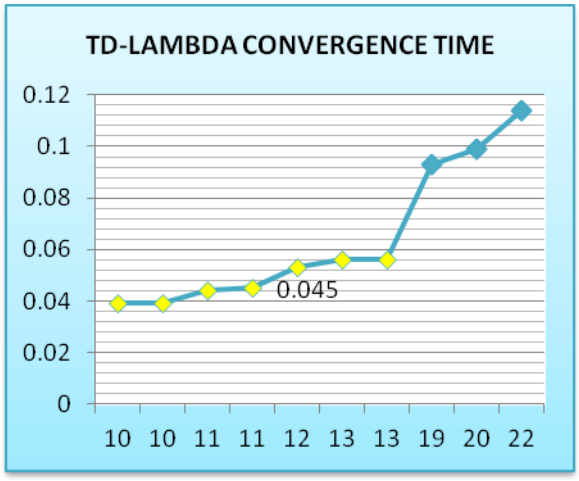
Appendix A4: SARSA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	2.377	12	0.057	489	33
2.	2.535	15	0.076	486	29
3.	2.671	22	0.118	479	33
4.	2.921	30	0.175	471	34
5.	2.707	22	0.119	479	31
6.	2.532	17	0.086	484	31
7.	2.351	10	0.047	491	33
8.	2.945	30	0.177	471	34
9.	2.779	23	0.128	478	33
10.	2.528	15	0.076	486	33



Appendix A5: TD(λ)

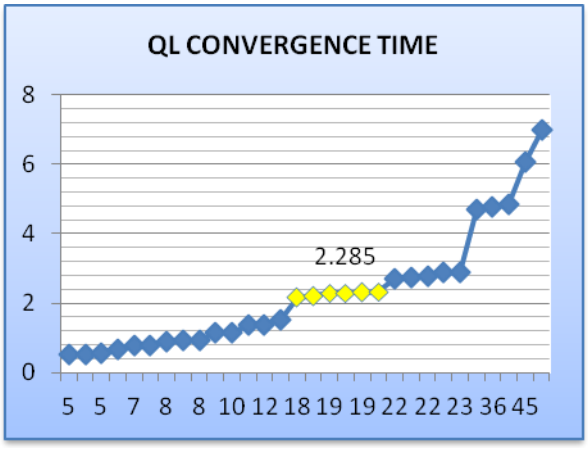
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	1.955	10	0.039	491	31
2.	2.602	22	0.114	479	31
3.	2.483	20	0.099	481	34
4.	2.156	13	0.056	488	33
5.	2.038	11	0.044	490	32
6.	2.039	11	0.045	490	33
7.	2.224	12	0.053	489	33
8.	1.971	10	0.039	491	34
9.	2.166	13	0.056	488	32
10.	2.448	19	0.093	482	32



APPENDIX B: Comparison between RL algorithms without restriction to the number of containers (1500x600).

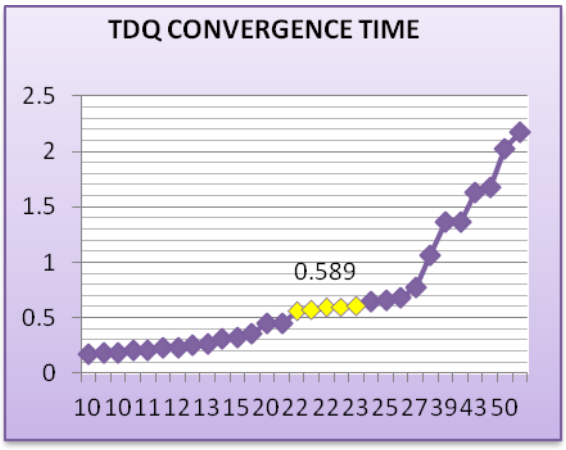
Appendix B1: Q-learning

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	56.171	8	0.899	493	63
2.	57.846	12	1.388	489	60
3.	53.441	5	0.534	496	63
4.	58.189	8	0.931	493	64
5.	58.248	13	1.514	488	65
6.	57.452	12	1.379	489	62
7.	60.046	19	2.282	482	60
8.	54.587	6	0.655	495	60
9.	67.489	45	6.074	456	62
10.	61.141	22	2.690	479	62
11.	58.089	8	0.929	493	64
12.	62.700	22	2.759	479	65
13.	59.820	18	2.154	483	65
14.	53.463	5	0.535	496	62
15	62.481	23	2.874	478	64
16.	55.813	7	0.781	494	61
17.	65.640	37	4.857	464	63
18.	60.132	19	2.285	482	62
19.	65.375	36	4.707	465	62
20.	57.264	10	1.145	491	65
21.	60.903	19	2.314	482	60
22.	62.366	22	2.744	479	65
23.	61.219	18	2.204	483	65
24.	54.806	5	0.548	496	62
25.	62.557	23	2.878	478	64
26.	56.030	7	0.784	494	61
27.	94.604	37	7.000	464	63
28.	61.127	19	2.323	482	62
29.	66.281	36	4.772	465	62
30.	57.397	10	1.148	491	65



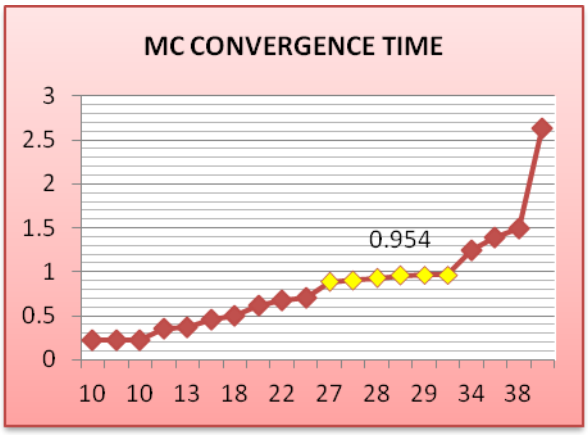
Appendix B2: TDQ

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	12.958	23	0.596	478	64
2.	10.534	15	0.316	486	64
3.	11.014	16	0.352	485	66
4.	20.930	52	2.177	449	62
5.	9.447	12	0.227	489	62
6.	9.536	12	0.229	489	62
7.	9.932	13	0.258	488	66
8.	17.451	39	1.361	462	64
9.	19.069	44	1.678	457	62
10.	9.780	13	0.254	488	64
11.	13.228	25	0.661	476	63
12.	9.025	10	0.181	491	64
13.	12.878	22	0.567	479	61
14.	10.464	15	0.314	486	62
15	11.868	19	0.451	482	61
16.	8.801	10	0.176	491	61
17.	13.110	23	0.603	478	62
18.	9.188	11	0.202	490	62
19.	14.271	27	0.771	474	61
20.	13.521	24	0.649	477	60
21.	13.651	25	0.683	476	63
22.	11.929	20	0.447	481	62
23.	20.281	50	2.028	451	63
24.	9.238	10	0.185	491	64
25.	12.663	22	0.557	479	61
26.	17.471	39	1.363	462	62
27.	16.028	33	1.058	468	62
28.	18.972	43	1.632	458	64
29.	13.396	22	0.589	478	62
30.	9.234	11	0.203	490	62



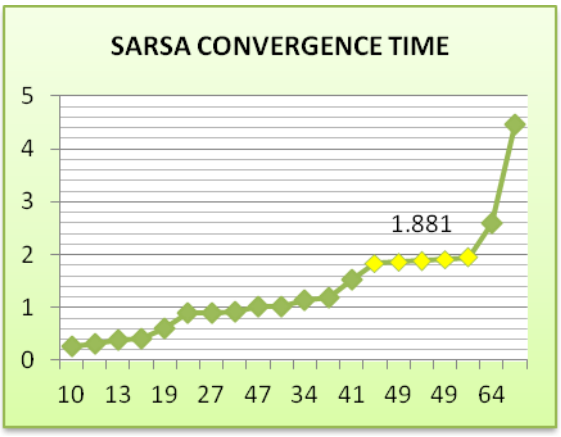
Appendix B3: Monte-Carlo

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	13.627	17	0.463	484	63
2.	13.757	18	0.495	483	65
3.	16.586	28	0.929	473	62
4.	13.898	13	0.361	488	64
5.	16.633	29	0.965	472	65
6.	10.995	10	0.220	491	64
7.	18.393	34	1.251	467	62
8.	15.346	23	0.706	478	64
9.	23.942	55	2.634	446	61
10.	14.725	21	0.618	480	65
11.	14.134	13	0.367	488	64
12.	16.730	27	0.903	474	64
13.	15.257	22	0.671	478	65
14.	10.855	10	0.217	491	62
15	17.028	28	0.954	473	64
16.	16.397	27	0.885	474	63
17.	19.584	38	1.488	463	62
18.	10.927	10	0.219	491	60
19.	19.351	36	1.393	465	65
20.	17.244	28	0.966	473	65



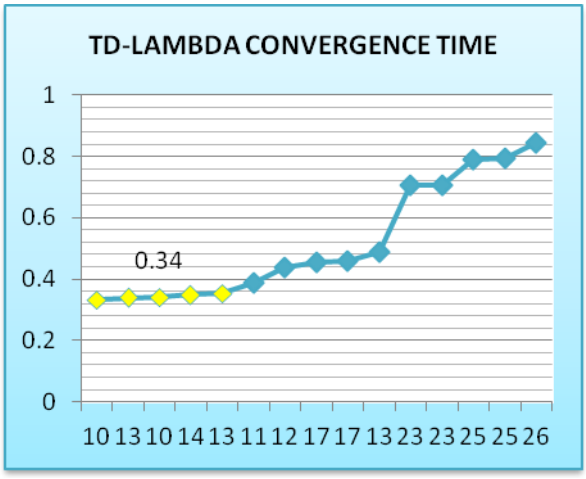
Appendix B4: SARSA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	20.366	64	2.607	437	64
2.	16.588	27	0.896	474	62
3.	16.301	28	0.913	473	62
4.	24.190	92	4.451	409	65
5.	10.763	47	1.012	464	64
6.	14.599	14	0.409	487	62
7.	18.807	50	1.881	451	62
8.	18.660	41	1.530	460	62
9.	16.861	34	1.147	467	62
10.	15.686	19	0.596	482	65
11.	19.530	49	1.914	452	63
12.	18.395	50	1.840	451	61
13.	15.980	28	0.895	473	64
14.	14.435	11	0.317	490	65
15	14.503	13	0.377	488	60
16.	13.782	10	0.276	491	64
17.	18.973	49	1.859	452	64
18.	17.356	34	1.180	467	65
19.	19.069	51	1.945	450	62
20.	17.470	29	1.013	472	64



Appendix B5: TD(λ)

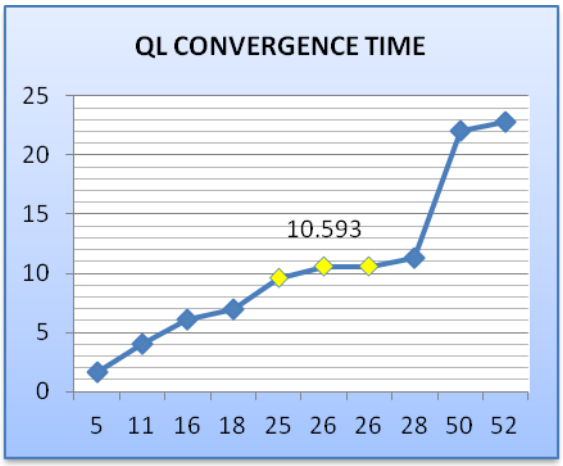
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	13.049	13	0.339	488	64
2.	13.389	17	0.455	484	62
3.	15.354	23	0.706	478	64
4.	15.844	25	0.792	476	62
5.	12.418	14	0.348	487	62
6.	16.231	26	0.844	475	62
7.	13.579	13	0.353	488	64
8.	13.433	17	0.457	484	62
9.	15.394	23	0.708	478	64
10.	15.865	25	0.793	476	62
11.	18.347	12	0.440	489	61
12.	16.979	10	0.340	491	60
13.	16.661	10	0.333	491	63
14.	17.552	11	0.386	490	61
15.	18.793	13	0.488	488	60



APPENDIX C: Comparison between RL algorithms without restriction to the number of containers (1500x900).

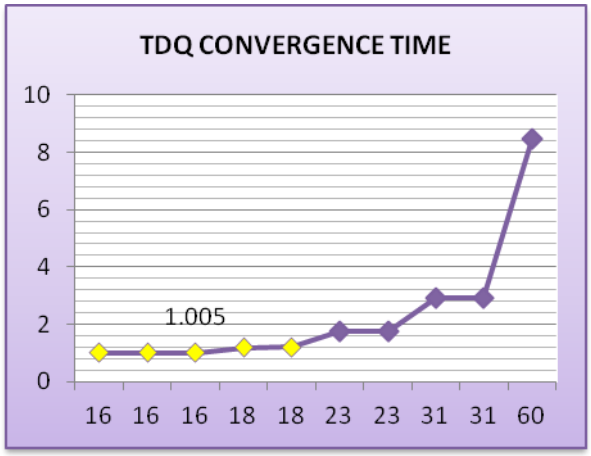
Appendix C1: Q-learning

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	203.778	26	10.596	475	93
2.	166.553	5	1.666	496	95
3.	201.788	28	11.300	473	95
4.	190.851	16	6.107	485	92
5.	203.716	26	10.593	475	93
6.	192.500	18	6.930	483	95
7.	184.413	11	4.057	490	92
8.	191.568	25	9.578	476	92
9.	219.562	52	22.834	449	92
10.	220.257	50	22.026	451	96



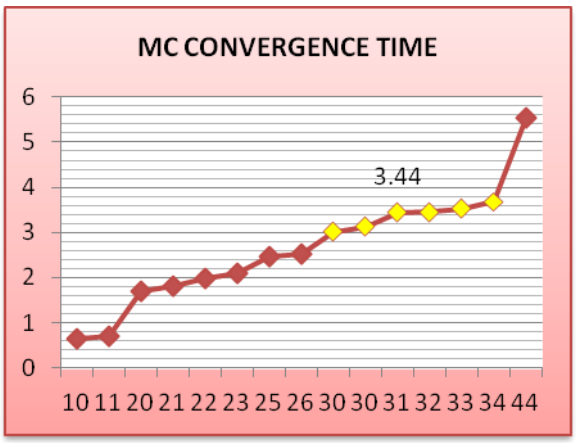
Appendix C2: TDQ

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	47.268	31	2.931	470	93
2.	37.895	23	1.743	478	94
3.	47.309	31	2.933	470	93
4.	38.016	23	1.749	478	95
5.	33.199	18	1.195	483	94
6.	70.517	60	8.462	441	91
7.	31.170	16	0.997	485	92
8.	32.940	18	1.186	483	95
9.	31.187	16	0.998	485	94
10.	31.403	16	1.005	485	95



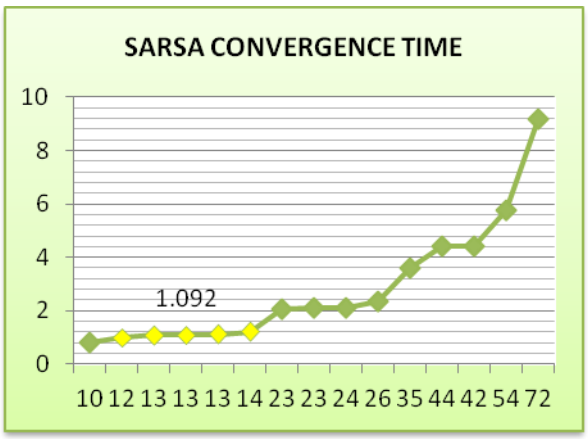
Appendix C3: Monte-Carlo

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	45.040	22	1.982	479	96
2.	53.870	32	3.448	469	93
3.	52.157	30	3.129	471	94
4.	32.344	10	0.647	491	94
5.	50.164	30	3.009	471	92
6.	55.483	31	3.440	470	93
7.	45.383	23	2.088	478	95
8.	48.597	26	2.527	475	93
9.	43.403	21	1.823	480	95
10.	42.362	20	1.694	481	93
11.	54.171	34	3.684	467	93
12.	32.154	11	0.707	490	93
13.	63.013	44	5.545	457	95
14.	53.339	33	3.520	468	97
15.	49.410	25	2.471	476	94



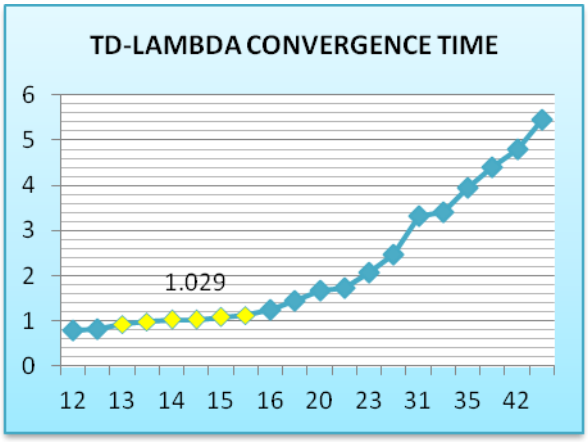
Appendix C4: SARSA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	41.624	12	0.999	489	95
2.	49.937	44	4.394	457	94
3.	42.663	13	1.109	488	92
4.	44.522	23	2.048	478	92
5.	44.360	24	2.129	477	94
6.	45.485	26	2.365	475	93
7.	43.308	14	1.213	487	93
8.	52.445	42	4.405	459	93
9.	41.389	13	1.076	488	96
10.	63.615	72	9.161	429	94
11.	51.267	35	3.589	466	95
12.	45.704	23	2.102	478	94
13.	39.804	10	0.796	491	95
14.	53.389	54	5.766	447	93
15.	41.988	13	1.092	488	96



Appendix C5: TD-LAMBDA

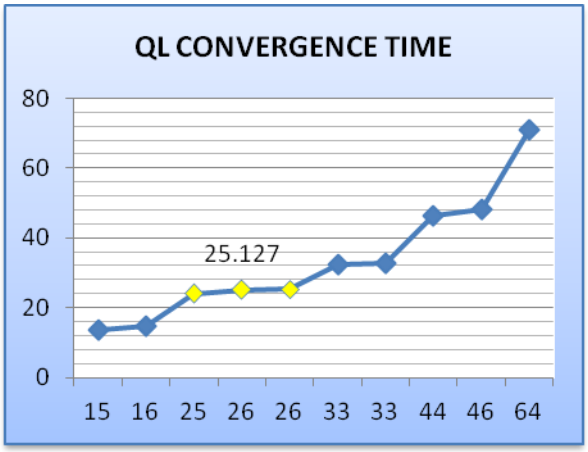
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	40.234	18	1.448	483	96
2.	34.326	12	0.824	489	93
3.	54.755	31	3.395	470	93
4.	44.994	23	2.070	478	95
5.	41.886	20	1.675	481	93
6.	42.892	20	1.716	481	93
7.	35.325	13	0.918	488	95
8.	53.677	31	3.328	470	94
9.	38.551	16	1.234	485	92
10.	34.987	14	0.980	487	93
11.	47.601	26	2.475	475	95
12.	32.305	12	0.775	489	94
13.	58.425	42	4.791	459	94
14.	63.513	43	5.462	458	95
15.	35.857	15	1.076	486	92
16.	36.493	14	1.022	487	95
17.	56.366	35	3.946	466	96
18.	36.748	14	1.029	487	97
19.	59.356	37	4.392	464	92
20.	37.267	15	1.118	486	94


TD-LAMBDA CONVERGENCE TIME
6
5
4
3
2
1
0
1.029
12
13
14
15
16
20
23
31
35
42

APPENDIX D: Comparison between RL algorithms without restriction to the number of containers (1500x1200).

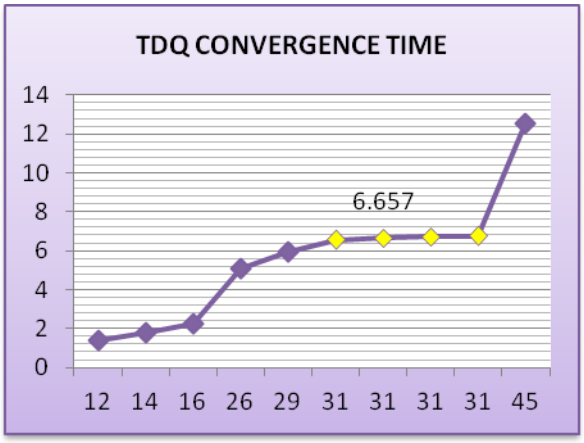
Appendix D1: Q-learning

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	480.463	25	24.023	476	127
2.	492.362	33	32.496	468	128
3.	452.529	15	13.576	486	125
4.	521.805	46	48.001	455	125
5.	497.836	33	32.857	468	124
6.	487.579	26	25.354	475	127
7.	553.058	64	70.791	437	126
8.	483.206	26	25.127	475	127
9.	457.812	16	14.650	485	124
10.	527.909	44	46.456	457	127



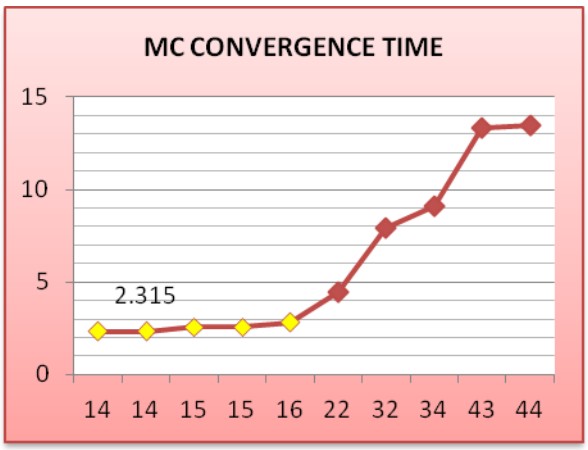
Appendix D2: TDQ

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	64.235	14	1.799	487	125
2.	107.374	31	6.657	470	126
3.	102.464	29	5.943	472	123
4.	139.199	45	12.528	456	124
5.	58.486	12	1.404	489	126
6.	105.829	31	6.561	470	125
7.	70.371	16	2.252	485	127
8.	108.864	31	6.750	470	127
9.	97.624	26	5.076	475	125
10.	108.327	31	6.716	470	127



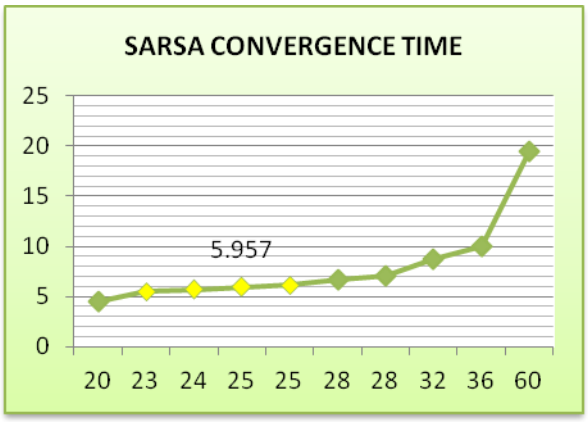
Appendix D3: Monte-Carlo

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	123.764	32	7.921	469	124
2.	133.812	34	9.099	467	127
3.	82.747	14	2.317	487	127
4.	155.066	43	13.336	458	125
5.	85.941	15	2.578	486	126
6.	88.094	16	2.819	485	126
7.	152.850	44	13.450	457	128
8.	100.720	22	4.432	479	125
9.	85.312	15	2.559	486	123
10.	82.661	14	2.315	487	126



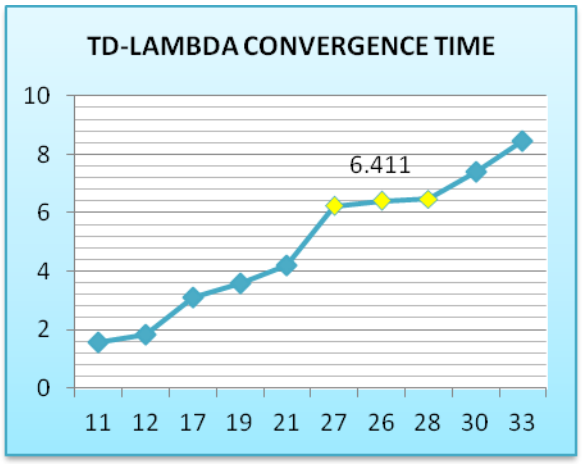
Appendix D4: SARSA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	118.548	28	6.639	473	128
2.	122.305	25	6.115	476	124
3.	119.876	23	5.514	478	124
4.	135.788	32	8.690	469	126
5.	139.530	36	10.046	465	125
6.	162.335	60	19.480	441	124
7.	111.778	20	4.471	481	126
8.	118.240	24	5.676	477	127
9.	119.149	25	5.957	476	126
10.	125.069	28	7.004	473	127



Appendix D5: TD-LAMBDA

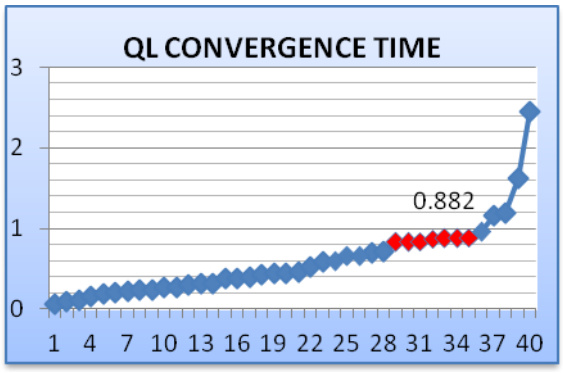
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL	NUMBER OF CONTAINER
1.	115.748	28	6.481	473	124
2.	123.296	30	7.398	471	126
3.	114.063	26	6.411	475	127
4.	100.286	21	4.212	480	126
5.	76.141	12	1.827	489	125
6.	128.255	33	8.465	468	125
7.	94.563	19	3.593	482	124
8.	91.123	17	3.098	484	124
9.	115.422	27	6.233	474	128
10.	72.045	11	1.585	490	124



APPENDIX E: Comparison between RL and EAs with restriction (1500x300).

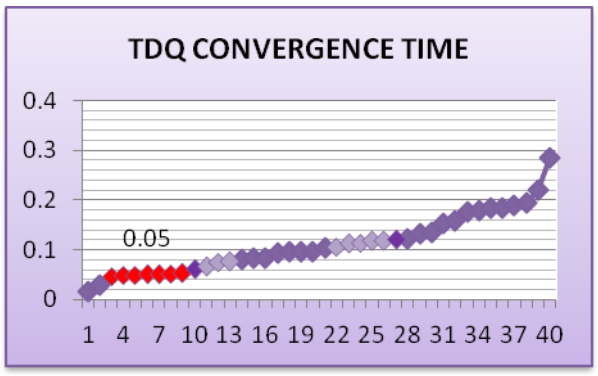
Appendix E1: Q-learning

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	7.253	58	0.841	443
2.	7.129	46	0.656	455
3.	6.840	20	0.274	481
4.	6.874	43	0.591	458
5.	6.577	5	0.066	496
6.	**6.511**	**8**	**0.104**	**493**
7.	6.557	17	0.223	484
8.	7.013	62	0.870	439
9.	7.035	85	1.196	416
10.	6.513	24	0.313	477
11.	6.651	33	0.439	468
12.	6.630	40	0.530	461
13.	6.937	52	0.721	449
14.	6.736	29	0.391	472
15.	7.049	59	0.832	442
16.	6.993	21	0.294	480
17.	6.854	28	0.384	473
18.	6.524	14	0.183	487
19.	7.177	81	1.163	420
20.	6.610	18	0.238	483
21.	7.200	58	0.835	443
22.	7.147	46	0.658	455
23.	6.858	20	0.274	481
24.	6.910	43	0.594	458
25.	6.393	8	0.102	493
26.	7.122	62	0.883	439
27.	6.506	24	0.312	477
28.	6.529	33	0.431	468
29.	6.683	52	0.695	449
30.	6.506	29	0.377	472
31.	7.033	63	0.887	438
32.	6.618	34	0.450	467
33.	6.443	12	0.155	489
34.	6.891	64	0.882	437
35.	6.902	33	0.456	468
36.	7.343	65	0.955	436
37.	6.406	16	0.205	485
38.	6.403	19	0.243	482
39.	7.911	155	2.452	346
40.	7.520	108	1.624	392



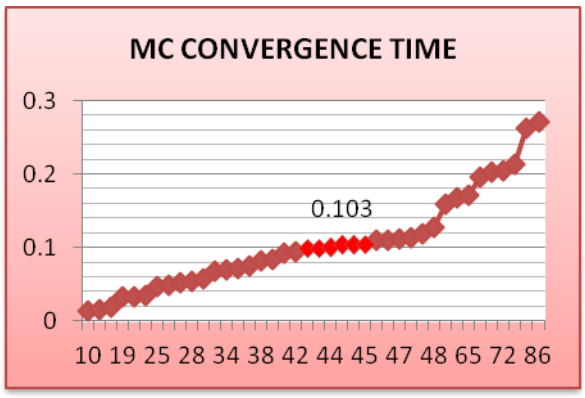
Appendix E2: TDQ

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	0.883	16	0.028	485
2.	1.089	43	0.094	458
3.	1.414	69	0.195	432
4.	1.099	44	0.097	457
5.	1.571	91	0.286	410
6.	1.284	60	0.154	441
7.	1.011	38	0.077	463
8.	1.416	78	0.221	423
9.	**0.694**	**11**	**0.015**	**490**
10.	1.197	56	0.134	445
11.	1.039	40	0.083	461
12.	1.111	47	0.104	454
13.	0.976	34	0.066	467
14.	1.344	70	0.188	431
15.	0.950	32	0.061	469
16.	0.905	28	0.051	473
17.	1.042	39	0.081	462
18.	1.051	40	0.084	461
19.	1.080	45	0.097	456
20.	1.323	67	0.177	434
21.	2.259	35	0.158	466
22.	0.911	29	0.053	472
23.	1.155	51	0.118	450
24.	1.153	51	0.118	450
25.	1.101	48	0.106	453
26.	0.902	26	0.047	474
27.	1.171	57	0.133	444
28.	0.909	27	0.049	474
29.	1.344	69	0.185	432
30.	0.989	38	0.075	463
31.	0.921	28	0.052	473
32.	1.204	51	0.122	450
33.	1.177	51	0.120	450
34.	1.186	48	0.114	453
35.	0.910	27	0.049	474
36.	1.167	48	0.112	453
37.	1.371	67	0.184	432
38.	2.490	36	0.179	465
39.	0.930	27	0.050	474
40.	1.075	45	0.097	456



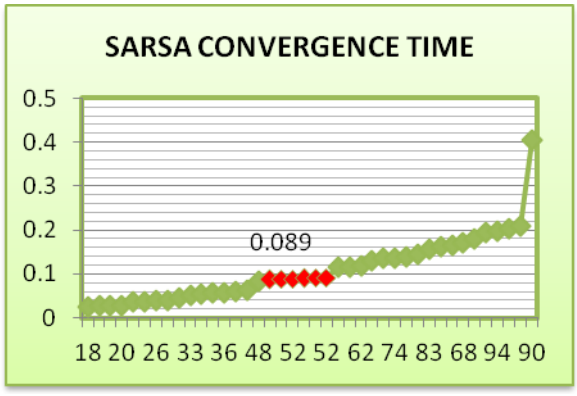
Appendix E3: Monte-Carlo

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	1.372	61	0.167	438
2.	1.038	34	0.071	467
3.	1.417	72	0.204	429
4.	**0.729**	**10**	**0.015**	**491**
5.	1.105	45	0.099	456
6.	1.391	70	0.195	431
7.	1.561	84	0.262	417
8.	1.577	86	0.271	415
9.	1.180	48	0.113	453
10.	1.409	72	0.203	429
11.	0.857	20	0.034	481
12.	1.121	42	0.094	459
13.	1.147	45	0.103	456
14.	1.166	47	0.110	454
15.	0.768	12	0.018	489
16.	1.142	45	0.103	456
17.	1.054	35	0.074	466
18.	1.136	44	0.100	457
19.	1.159	45	0.104	456
20.	0.986	29	0.057	472
21.	1.335	48	0.128	453
22.	0.855	19	0.032	482
23.	1.076	39	0.084	462
24.	0.954	27	0.052	474
25.	1.315	65	0.171	436
26.	1.343	59	0.158	442
27.	0.724	10	0.014	491
28.	1.445	74	0.214	427
29.	0.933	26	0.049	475
30.	1.014	33	0.067	468
31.	1.124	41	0.092	460
32.	0.959	28	0.054	473
33.	1.213	49	0.119	452
34.	1.165	47	0.110	454
35.	0.920	25	0.046	476
36.	1.064	38	0.081	463
37.	1.148	43	0.099	458
38.	1.035	34	0.070	467
39.	1.177	47	0.111	454
40.	0.847	19	0.032	482



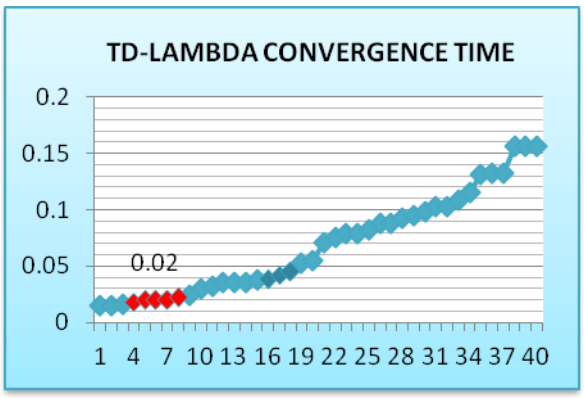
Appendix E4: SARSA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	2.250	90	0.405	411
2.	1.013	89	0.180	412
3.	0.969	72	0.140	429
4.	0.925	62	0.115	439
5.	1.053	94	0.198	407
6.	0.798	37	0.059	464
7.	0.765	26	0.040	475
8.	1.009	80	0.161	421
9.	0.754	24	0.036	477
10.	0.701	20	0.028	481
11.	0.952	68	0.129	433
12.	0.852	52	0.089	449
13.	0.797	36	0.057	465
14.	1.035	94	0.195	407
15.	0.901	64	0.115	437
16.	0.718	18	0.026	483
17.	0.947	72	0.136	429
18.	**0.663**	**13**	**0.028**	**488**
19.	1.076	97	0.209	404
20.	0.778	33	0.051	468
21.	0.866	32	0.055	469
22.	0.879	51	0.090	450
23.	0.929	74	0.137	427
24.	0.765	30	0.046	471
25.	0.871	52	0.091	449
26.	0.792	36	0.057	465
27.	0.891	49	0.087	452
28.	0.867	48	0.083	453
29.	0.935	83	0.155	418
30.	0.855	52	0.089	449
31.	1.136	89	0.202	412
32.	1.002	72	0.144	429
33.	0.957	62	0.119	439
34.	0.861	37	0.064	464
35.	0.792	26	0.041	475
36.	1.023	80	0.164	421
37.	0.742	24	0.036	477
38.	0.699	20	0.028	481
39.	1.258	68	0.171	433
40.	0.883	52	0.092	449



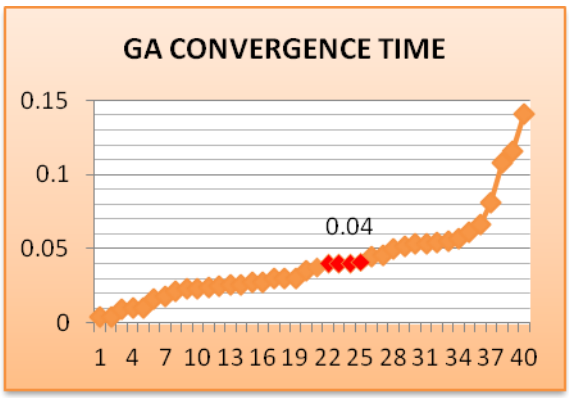
Appendix E5: TD-LAMBDA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	0.820	18	0.030	483
2.	0.768	12	0.018	489
3.	1.114	44	0.098	457
4.	0.904	25	0.045	476
5.	1.097	40	0.088	461
6.	1.110	43	0.095	458
7.	1.217	54	0.131	447
8.	1.220	54	0.132	447
9.	0.771	13	0.020	488
10.	1.357	38	0.103	463
11.	1.021	35	0.071	466
12.	1.576	94	0.296	407
13.	**0.731**	**10**	**0.015**	**491**
14.	1.145	50	0.115	451
15.	1.123	46	0.103	455
16.	1.022	11	0.020	490
17.	1.033	37	0.076	464
18.	1.318	67	0.177	434
19.	1.067	43	0.092	458
20.	1.321	59	0.156	442
21.	1.323	67	0.177	434
22.	0.952	29	0.055	472
23.	0.886	20	0.035	481
24.	0.731	10	0.015	491
25.	0.943	28	0.053	473
26.	0.872	22	0.038	479
27.	0.789	15	0.024	486
28.	1.036	38	0.079	463
29.	1.069	41	0.088	460
30.	1.149	47	0.108	454
31.	0.838	19	0.032	482
32.	0.761	13	0.020	488
33.	1.066	37	0.079	464
34.	1.075	38	0.082	463
35.	0.861	21	0.036	480
36.	0.862	22	0.038	479
37.	0.789	14	0.022	487
38.	0.748	11	0.016	490
39.	0.871	20	0.035	481
40.	0.905	23	0.042	478



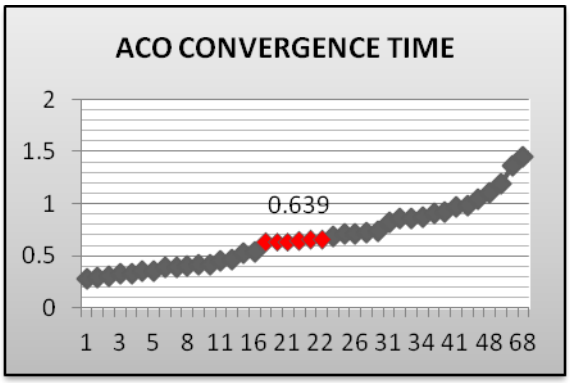
Appendix E6: GA

S/N	SIMULATION TIME	CONVERGENCE GENERATION	CONVERGENCE TIME
1.	4.229	3	0.025
2.	3.541	8	0.057
3.	3.349	4	0.027
4.	3.314	8	0.053
5.	2.612	2	0.010
6.	2.402	2	0.010
7.	3.343	8	0.053
8.	3.400	8	0.054
9.	3.019	5	0.030
10.	3.050	9	0.055
11.	3.434	4	0.027
12.	3.678	9	0.066
13.	3.063	6	0.037
14.	3.283	4	0.026
15.	3.049	3	0.018
16.	4.910	11	0.108
17.	2.985	5	0.030
18.	3.404	17	0.116
19.	**2.204**	**1**	**0.004**
20.	2.593	5	0.026
21.	2.967	4	0.024
22.	3.274	8	0.052
23.	2.570	4	0.021
24.	2.897	14	0.081
25.	3.730	6	0.045
26.	2.916	6	0.035
27.	2.005	1	0.004
28.	2.907	7	0.041
29.	2.315	2	0.009
30.	2.653	3	0.016
31.	3.301	6	0.040
32.	3.298	6	0.040
33.	2.334	13	0.061
34.	3.155	8	0.050
35.	2.820	4	0.023
36.	3.315	9	0.046
37.	2.855	7	0.040
38.	2.941	24	0.141
39.	2.273	5	0.023
40.	2.992	5	0.030



Appendix E7: ACO

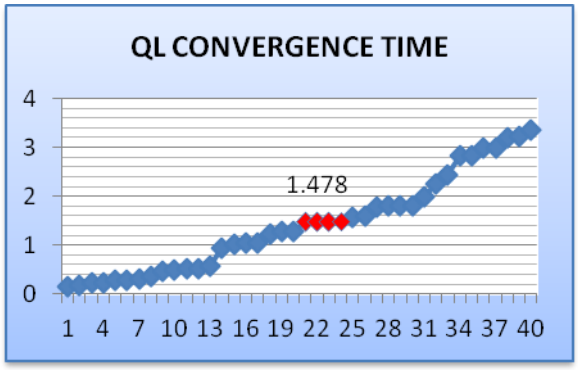
S/N	CONVERGENCE TIME	CONVERGENCE GENERATION
1.	0.821	31
2.	0.325	3
3.	0.639	22
4.	0.741	27
5.	0.714	25
6.	0.394	7
7.	0.468	12
8.	0.284	1
9.	0.529	15
10.	0.636	22
11.	0.654	22
12.	0.388	7
13.	1.360	63
14.	0.686	25
15.	0.420	9
16.	0.919	37
17.	0.354	5
18.	1.049	45
19.	**0.458**	**11**
20.	0.876	34
21.	0.293	2
22.	0.536	16
23.	0.355	5
24.	0.650	22
25.	0.858	34
26.	0.405	8
27.	0.329	4
28.	0.637	21
29.	0.973	41
30.	0.855	35
31.	0.727	25
32.	0.300	2
33.	0.978	41
34.	1.104	48
35.	1.448	68
36.	0.418	9
37.	0.717	26
38.	0.632	21
39.	1.186	54
40.	0.912	37



APPENDIX F: Comparison between RL and Eas with restriction (1500x600).

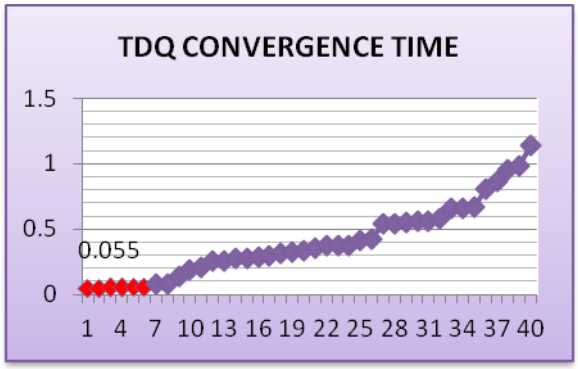
Appendix F1: Q-learning

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	18.132	41	1.487	460
2.	14.661	5	0.147	496
3.	17.852	50	1.785	451
4.	16.791	30	1.008	471
5.	17.766	44	1.563	457
6.	15.382	10	0.308	491
7.	17.852	36	1.285	465
8.	17.838	51	1.819	450
9.	19.650	82	3.223	419
10.	16.609	28	0.930	473
11.	18.187	55	2.001	446
12.	15.518	9	0.279	492
13.	19.195	74	2.840	427
14.	19.467	77	2.998	424
15.	15.106	6	0.181	495
16.	15.733	15	0.472	486
17.	15.921	16	0.509	485
18.	15.992	18	0.576	483
19.	17.255	37	1.277	464
20.	19.531	82	3.203	419
21.	19.424	63	2.447	438
22.	19.971	75	2.996	426
23.	18.082	50	1.808	451
24.	18.151	50	1.815	451
25.	16.410	15	0.492	486
26.	16.945	31	1.051	470
27.	17.824	45	1.604	456
28.	17.593	42	1.478	459
29.	15.078	8	0.241	493
30.	17.548	42	1.474	459
31.	15.315	8	0.245	493
32.	19.796	85	3.365	416
33.	19.194	74	2.841	427
34.	15.601	12	0.374	489
35.	18.457	61	2.252	440
36.	16.866	31	1.046	470
37.	17.595	42	1.478	459
38.	17.090	36	1.230	465
39.	15.998	16	0.512	485
40.	15.426	9	0.278	492



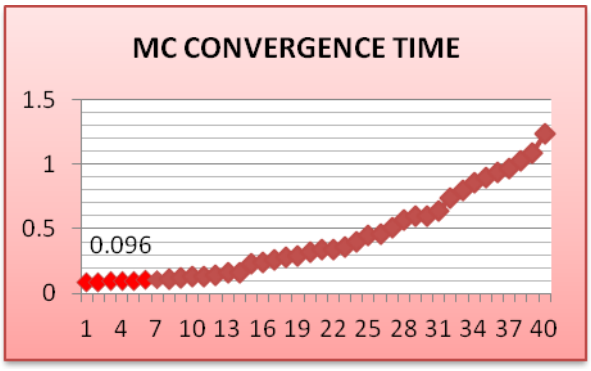
Appendix F2: TDQ

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	5.400	62	0.670	439
2.	2.437	10	0.049	491
3.	4.627	46	0.426	455
4.	2.748	14	0.077	487
5.	4.487	42	0.377	459
6.	4.472	42	0.376	459
7.	6.329	75	0.949	426
8.	5.083	55	0.559	446
9.	4.363	41	0.358	460
10.	5.479	60	0.657	441
11.	4.443	38	0.338	463
12.	6.331	78	0.988	423
13.	2.489	11	0.055	490
14.	3.630	28	0.203	473
15.	5.398	61	0.659	440
16.	5.944	72	0.863	429
17.	4.551	45	0.410	456
18.	2.438	10	0.049	491
19.	3.245	22	0.143	479
20.	5.098	55	0.561	446
21.	4.156	33	0.274	468
22.	2.524	11	0.056	490
23.	2.733	14	0.077	487
24.	5.122	53	0.543	448
25.	4.078	36	0.294	465
26.	2.545	11	0.056	490
27.	6.705	85	1.140	416
28.	5.155	56	0.577	445
29.	3.918	33	0.259	468
30.	4.046	34	0.275	467
31.	4.496	42	0.378	459
32.	5.062	53	0.537	448
33.	4.175	38	0.317	463
34.	5.872	69	0.810	432
35.	3.932	33	0.260	468
36.	4.091	35	0.286	466
37.	2.514	11	0.055	490
38.	5.115	54	0.552	447
39.	4.183	39	0.326	462
40.	3.543	26	0.184	475



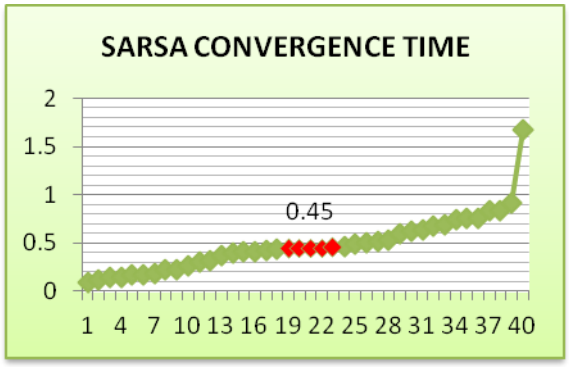
Appendix F3: Monte-Carlo

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	3.538	19	0.134	482
2.	3.225	16	0.103	485
3.	6.130	65	0.797	436
4.	6.858	79	1.084	422
5.	3.123	14	0.087	487
6.	3.464	19	0.132	482
7.	3.183	15	0.095	486
8.	4.977	45	0.448	456
9.	5.513	54	0.595	447
10.	5.428	52	0.565	449
11.	5.555	54	0.600	447
12.	4.612	37	0.341	464
13.	4.697	38	0.357	463
14.	3.388	18	0.140	483
15.	4.738	42	0.398	459
16.	4.352	34	0.296	467
17.	3.391	18	0.122	483
18.	5.972	62	0.740	439
19.	3.131	14	0.088	487
20.	5.197	49	0.509	452
21.	3.446	17	0.117	484
22.	4.160	29	0.241	472
23.	3.256	15	0.098	486
24.	5.079	45	0.457	456
25.	6.737	76	1.024	425
26.	3.291	17	0.112	484
27.	5.607	57	0.639	444
28.	4.071	29	0.236	472
29.	3.207	15	0.096	486
30.	6.379	70	0.893	431
31.	4.739	39	0.340	462
32.	6.657	73	0.972	428
33.	3.647	22	0.160	479
34.	6.454	73	0.942	428
35.	6.490	66	0.857	435
36.	4.213	31	0.261	470
37.	3.698	22	0.163	479
38.	7.161	86	1.232	415
39.	4.474	36	0.322	465
40.	4.289	33	0.282	468



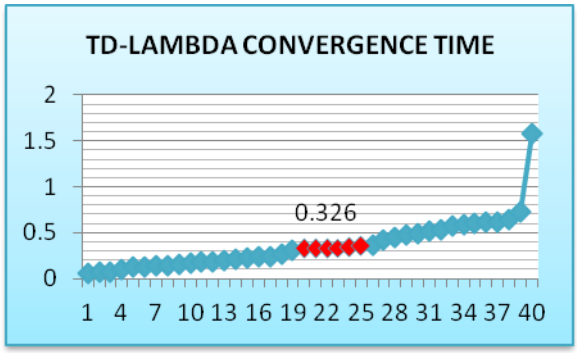
Appendix F4: SARSA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	3.744	19	0.142	482
2.	4.801	47	0.451	454
3.	4.517	48	0.434	453
4.	4.500	44	0.396	457
5.	3.878	24	0.186	477
6.	6.656	126	1.677	375
7.	4.608	41	0.378	460
8.	5.209	71	0.740	430
9.	4.703	52	0.489	449
10.	3.878	29	0.225	472
11.	4.186	36	0.301	465
12.	3.456	13	0.090	488
13.	4.578	48	0.439	453
14.	5.335	79	0.843	422
15.	3.642	16	0.117	485
16.	4.710	54	0.509	447
17.	4.890	43	0.421	458
18.	4.934	65	0.641	436
19.	5.473	77	0.843	424
20.	4.225	37	0.313	464
21.	4.356	47	0.409	454
22.	4.202	31	0.261	470
23.	5.112	68	0.695	433
24.	5.111	74	0.756	427
25.	4.022	22	0.177	479
26.	4.225	27	0.228	474
27.	5.036	67	0.675	434
28.	4.924	52	0.512	449
29.	5.408	85	0.919	416
30.	4.557	49	0.447	452
31.	5.249	72	0.756	429
32.	4.953	60	0.594	441
33.	4.544	46	0.418	455
34.	3.892	18	0.140	483
35.	4.829	65	0.628	436
36.	4.635	50	0.464	451
37.	4.648	49	0.456	452
38.	3.871	23	0.178	478
39.	4.790	55	0.527	446
40.	4.683	48	0.450	453



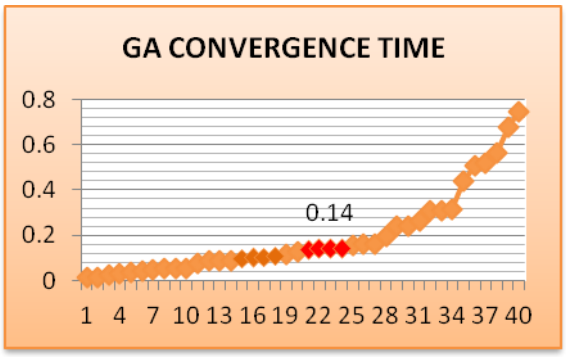
Appendix F5: TD-LAMBDA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	4.072	28	0.228	473
2.	3.507	20	0.140	481
3.	4.955	45	0.446	456
4.	5.869	55	0.646	446
5.	5.522	56	0.618	445
6.	4.613	39	0.360	462
7.	4.429	35	0.310	466
8.	4.323	28	0.242	473
9.	5.226	50	0.523	451
10.	4.913	43	0.423	458
11.	4.524	36	0.326	465
12.	5.910	62	0.733	439
13.	4.511	36	0.325	465
14.	3.795	25	0.190	476
15.	4.575	37	0.339	464
16.	4.543	37	0.336	464
17.	3.778	24	0.181	477
18.	5.550	54	0.599	447
19.	2.845	10	0.057	491
20.	5.670	54	0.612	447
21.	3.098	12	0.074	489
22.	5.567	53	0.590	448
23.	3.990	26	0.207	475
24.	3.746	21	0.157	480
25.	7.997	99	1.583	402
26.	3.737	23	0.172	478
27.	5.418	53	0.574	448
28.	5.071	48	0.487	453
29.	5.305	50	0.531	451
30.	3.555	20	0.142	481
31.	4.662	38	0.354	463
32.	5.126	47	0.482	454
33.	4.157	29	0.241	472
34.	3.280	16	0.105	485
35.	3.017	12	0.072	489
36.	4.322	31	0.268	470
37.	3.840	25	0.192	476
38.	3.429	18	0.123	483
39.	3.526	20	0.130	481
40.	4.517	36	0.325	465



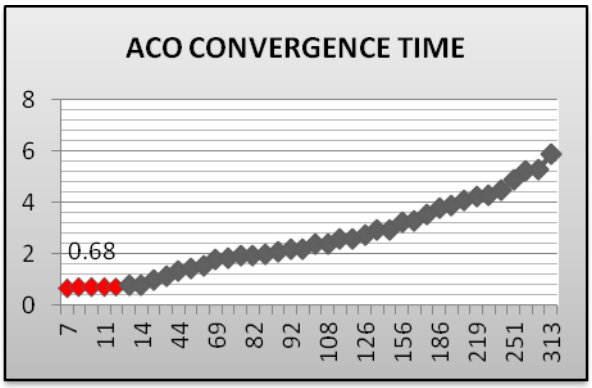
Appendix F6: GA

S/N	SIMULATION TIME	CONVERGENCE GENERATION	CONVERGENCE TIME
1.	4.571	5	0.046
2.	5.575	7	0.078
3.	5.573	12	0.134
4.	7.576	16	0.242
5.	5.538	8	0.089
6.	5.649	5	0.056
7.	5.659	9	0.102
8.	5.436	8	0.087
9.	5.707	17	0.194
10.	3.459	5	0.035
11.	12.132	18	0.437
12.	5.420	9	0.098
13.	3.364	2	0.013
14.	8.953	29	0.519
15.	8.978	9	0.162
16.	4.300	5	0.043
17.	9.346	14	0.262
18.	7.032	17	0.239
19.	4.867	12	0.117
20.	4.373	16	0.140
21.	9.156	37	0.678
22.	5.259	12	0.126
23.	5.371	13	0.140
24.	6.797	8	0.109
25.	9.235	17	0.314
26.	7.291	21	0.306
27.	3.277	2	0.013
28.	8.518	33	0.562
29.	12.745	20	0.510
30.	8.757	22	0.033
31.	5.560	28	0.311
32.	4.613	17	0.157
33.	5.331	5	0.053
34.	6.457	8	0.103
35.	6.944	10	0.139
36.	5.383	8	0.086
37.	6.129	13	0.159
38.	3.321	8	0.053
39.	7.017	53	0.744
40.	3.536	4	0.028



Appendix F7: ACO

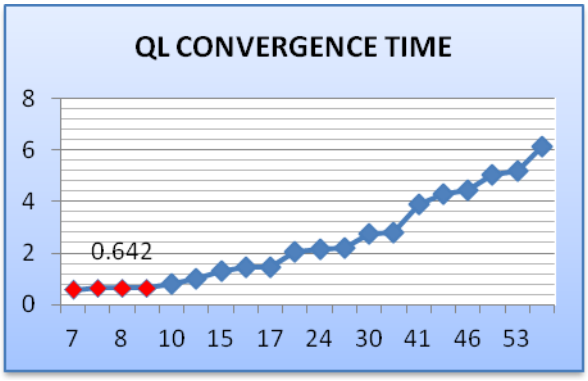
S/N	CONVERGENCE TIME	CONVERGENCE GENERATION
1.	4.856	251
2.	1.766	69
3.	1.914	81
4.	5.256	273
5.	2.717	126
6.	0.716	7
7.	3.752	186
8.	4.245	219
9.	0.771	14
10.	5.226	276
11.	2.579	119
12.	2.172	89
13.	4.059	205
14.	3.291	158
15.	2.578	118
16.	3.502	176
17.	4.480	230
18.	1.839	78
19.	**5.882**	**313**
20.	2.348	106
21.	1.402	52
22.	3.876	195
23.	2.382	108
24.	3.209	156
25.	1.975	84
26.	2.899	138
27.	1.333	44
28.	0.970	26
29.	1.121	34
30.	4.282	213
31.	0.699	11
32.	2.899	136
33.	2.049	88
34.	1.942	82
35.	1.542	57
36.	0.680	9
37.	0.746	13
38.	0.649	7
39.	0.673	9
40.	2.164	92



APPENDIX G: Tables of comparison between RL and Eas with restriction (1500x900).

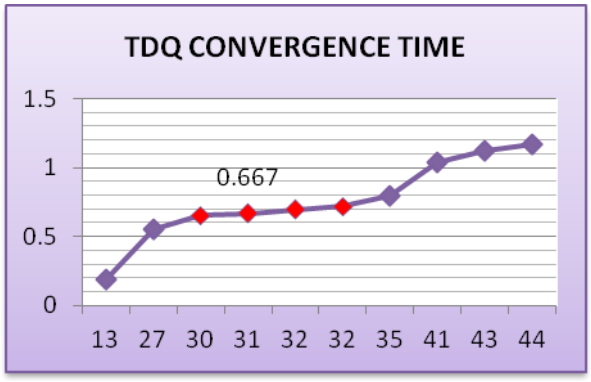
Appendix G1: Q-learning

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	39.930	8	0.639	493
2.	41.539	10	0.831	491
3.	44.095	25	2.205	476
4.	48.251	46	4.439	455
5.	59.997	12	1.440	489
6.	**47.159**	**41**	**3.867**	**460**
7.	49.137	53	5.209	448
8.	51.252	60	6.150	441
9.	40.568	8	0.649	493
10.	44.132	23	2.030	478
11.	49.518	51	5.051	450
12.	42.476	17	1.444	484
13.	45.438	31	2.817	470
14.	44.465	24	2.134	477
15.	42.725	15	1.282	486
16.	41.390	12	0.993	489
17.	47.767	45	4.299	456
18.	40.149	8	0.642	493
19.	40.102	7	0.561	494
20.	45.378	30	2.723	471



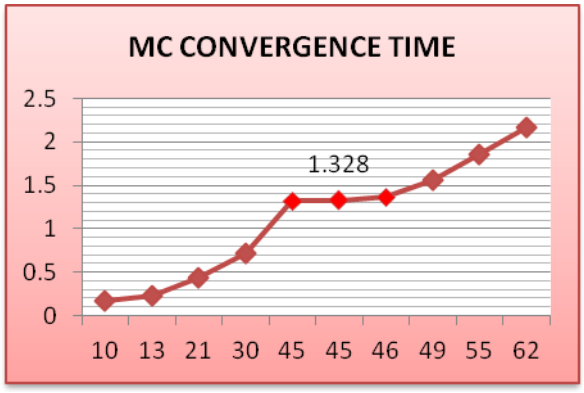
Appendix G2: TDQ

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	13.299	44	1.170	457
2.	11.381	35	0.797	466
3.	13.038	43	1.121	458
4.	10.139	27	0.548	474
5.	7.111	13	0.185	488
6.	**10.833**	**32**	**0.693**	**469**
7.	10.873	30	0.652	471
8.	12.661	41	1.038	460
9.	10.766	31	0.667	470
10.	11.204	32	0.717	469



Appendix G3: Monte-Carlo

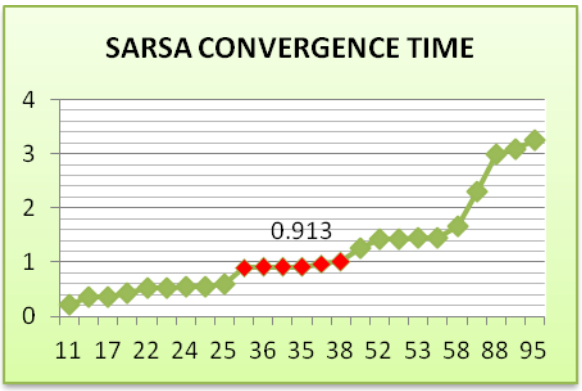
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	17.409	62	2.159	439
2.	8.266	10	0.165	491
3.	14.651	45	1.319	456
4.	14.818	46	1.363	455
5.	8.544	13	0.222	488
6.	**16.866**	**55**	**1.855**	**446**
7.	14.750	45	1.328	456
8.	15.875	49	1.556	452
9.	10.467	21	0.440	480
10.	11.981	30	0.719	471



Appendix G4: SARSA

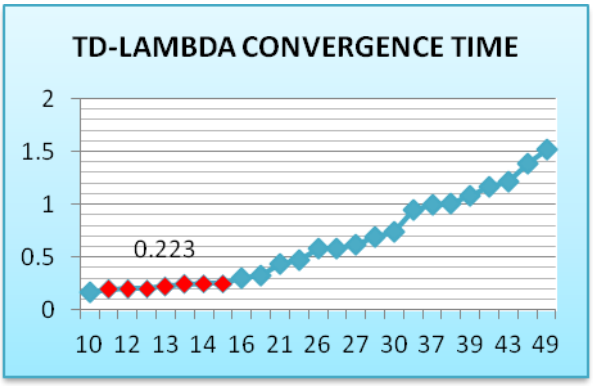
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	17.549	88	3.089	413
2.	11.693	22	0.514	479
3.	13.728	52	1.428	449
4.	11.532	24	0.554	477
5.	10.515	17	0.358	484
6.	**15.370**	**75**	**2.306**	**426**
7.	14.373	58	1.667	443
8.	17.161	95	3.261	406
9.	10.013	11	0.220	490
10.	11.352	19	0.431	482
11.	11.677	22	0.514	479
12.	13.717	52	1.427	449
13.	11.632	24	0.558	477
14.	10.706	17	0.364	484
15.	12.573	36	0.905	465
11.	12.009	25	0.600	476

12.	13.007	35	0.910	466
13.	13.036	35	0.913	466
14.	13.575	53	1.439	448
15.	16.975	88	2.988	413
16.	14.267	51	1.455	450
17.	14.318	44	1.260	457
18.	12.854	35	0.900	466
19.	13.305	38	1.011	463
20.	13.090	37	0.969	464



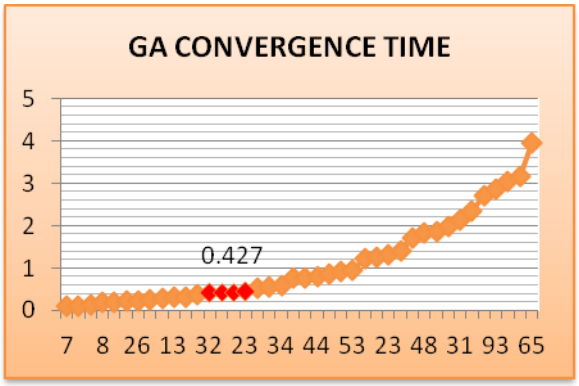
Appendix G5: TD-LAMBDA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	11.264	26	0.586	475
2.	11.078	26	0.576	475
3.	13.127	36	0.945	465
4.	11.560	27	0.624	474
5.	15.046	46	1.384	455
6.	**8.767**	**14**	**0.245**	**487**
7.	8.285	12	0.199	489
8.	8.579	13	0.223	488
9.	13.479	37	0.997	464
10.	14.166	43	1.218	458
11.	8.069	10	0.162	491
12.	9.392	16	0.301	485
13.	11.806	29	0.685	472
14.	8.483	12	0.204	489
15.	10.621	22	0.467	479
11.	14.268	41	1.170	460
12.	8.900	14	0.249	487
13.	13.320	38	1.012	463
14.	10.386	21	0.436	480
15.	15.445	49	1.514	452
16.	8.269	12	0.198	489
17.	9.585	17	0.326	484
18.	8.856	14	0.248	487
19	13.830	39	1.079	462
20.	12.386	30	0.743	471



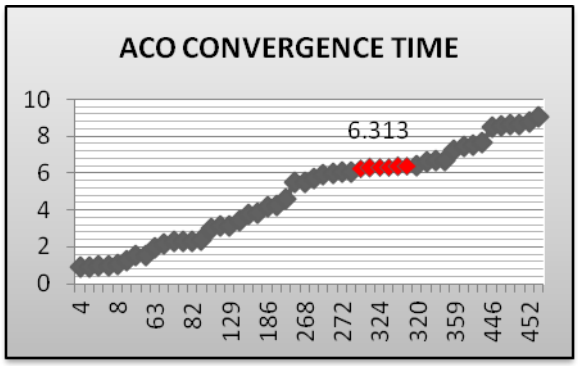
Appendix G6: GA

S/N	SIMULATION TIME	CONVERGENCE GENERATION	CONVERGENCE TIME
1.	23.847	36	1.717
2.	15.362	30	0.922
3.	31.559	48	3.030
4.	30.461	65	3.960
5.	18.483	50	1.848
6.	11.195	12	0.269
7.	8.659	18	0.312
8.	11.219	8	0.180
9.	11.830	13	0.308
10.	6.358	7	0.089
11.	19.094	48	1.833
12.	15.338	40	1.227
13.	34.368	31	2.131
14.	9.056	53	0.960
15.	12.284	31	0.762
16.	9.669	23	0.445
17.	59.140	23	2.720
18.	11.312	16	0.362
19.	6.476	33	0.427
20.	14.634	9	0.263
21.	28.382	23	1.306
22.	15.430	93	2.870
23.	15.705	24	0.754
24.	19.000	37	1.406
25.	12.615	22	0.555
26.	4.535	26	0.236
27.	11.681	10	0.234
28.	4.348	59	0.513
29.	6.220	11	0.137
30.	14.963	29	0.868
31.	8.775	34	0.597
32.	9.644	22	0.424
33.	4.609	22	0.203
34.	8.786	6	0.105
35.	9.095	44	0.800
36.	20.160	31	1.250
37.	36.491	32	2.335
38.	12.010	83	1.994
39.	19.866	80	3.178
40.	6.432	32	0.412



Appendix G7: ACO

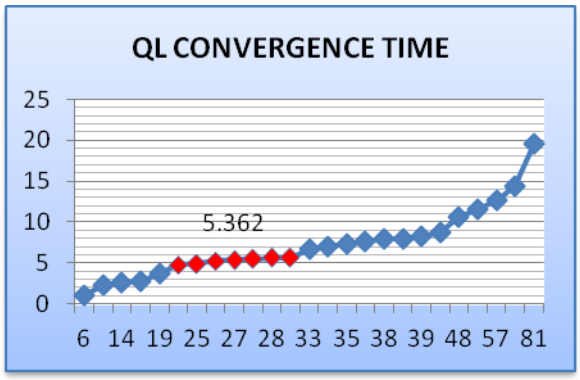
S/N	CONVERGENCE TIME	CONVERGENCE GENERATION
1.	6.007	304
2.	6.706	341
3.	2.162	42
4.	4.583	172
5.	7.456	379
6.	4.148	186
7.	2.302	82
8.	8.580	438
9.	5.927	287
10.	3.781	168
11.	6.315	314
12.	8.649	448
13.	7.552	387
14.	6.599	336
15.	1.008	4
16.	3.106	129
17.	6.372	321
18.	2.299	83
19.	**5.535**	**268**
20.	5.685	281
21.	9.055	474
22.	0.972	7
23.	6.081	308
24.	3.412	149
25.	8.658	458
26.	5.531	252
27.	8.516	446
28.	7.250	359
29.	2.999	125
30.	1.565	41
31.	6.309	299
32.	6.399	321
33.	8.762	452
34.	1.075	8
35.	1.515	38
36.	1.974	63
37.	6.442	320
38.	6.031	272
39.	0.941	3
40.	2.403	89
41	7.664	379
42	4.236	186
43	2.308	82
44	3.127	129
45	6.243	287
46	3.865	168
47	6.313	324
48	0.926	4
49	1.234	83
50	6.678	336


ACO CONVERGENCE TIME
10
8
6
4
2
0
6.313
4
8
63
82
129
186
268
272
324
320
359
446
452

APPENDIX H: Tables of comparison between RL and EAs with restriction (1500x1200).

Appendix H1: Q-learning

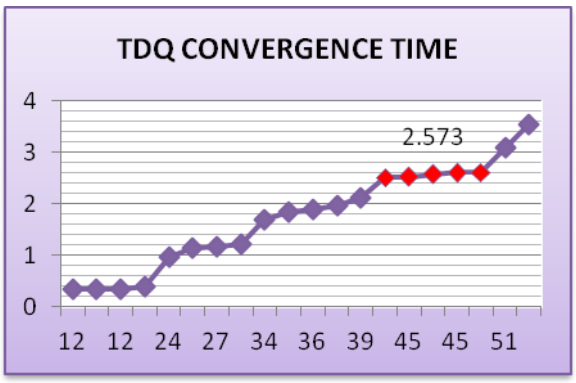
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	120.785	81	19.567	420
2.	104.749	38	7.961	463
3.	109.030	53	11.557	448
4.	114.251	63	14.396	438
5.	97.771	25	4.889	476
6.	95.645	25	4.782	476
7.	100.001	28	5.600	473
8.	101.653	33	6.709	468
9.	103.682	34	7.050	467
10.	87.573	6	1.051	495
11.	93.692	15	2.811	486
12.	111.660	57	12.729	444
13.	102.789	37	7.606	464
14.	104.523	42	8.780	459
15.	104.615	38	7.951	463
16.	93.511	12	2.244	489
17.	100.585	26	5.230	475
18.	100.455	28	5.625	473
19.	111.293	48	10.684	453
20.	96.511	19	3.667	482
21.	100.690	27	5.437	474
22.	99.291	27	5.362	474
23.	103.820	35	7.267	466
24.	94.040	14	2.633	487
25.	105.372	39	8.219	462



Appendix H2: TDQ

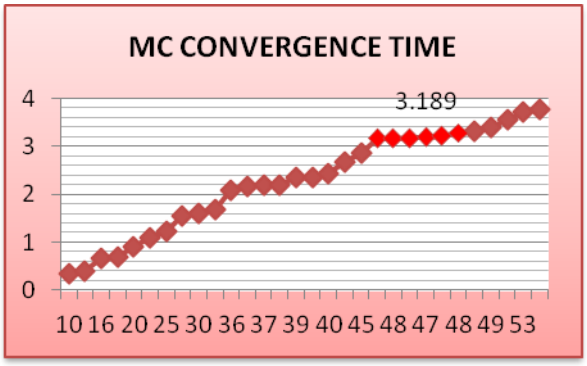
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	14.552	12	0.349	489
2.	27.121	36	1.953	465
3.	25.522	36	1.838	465
4.	28.887	45	2.600	456
5.	27.967	45	2.517	456
6.	**30.246**	**51**	**3.085**	**450**
7.	32.628	54	3.524	447
8.	28.448	44	2.503	457
9.	19.985	24	0.959	477
10.	24.802	34	1.687	467
11.	22.557	27	1.218	474
12.	26.311	36	1.894	465
13.	21.616	27	1.167	474
14.	26.942	39	2.101	462
15.	21.934	26	1.141	475
16.	15.351	13	0.399	488

17.	14.609	12	0.351	489
18.	28.590	45	2.573	456
19.	14.524	12	0.349	489
20.	28.970	45	2.607	456



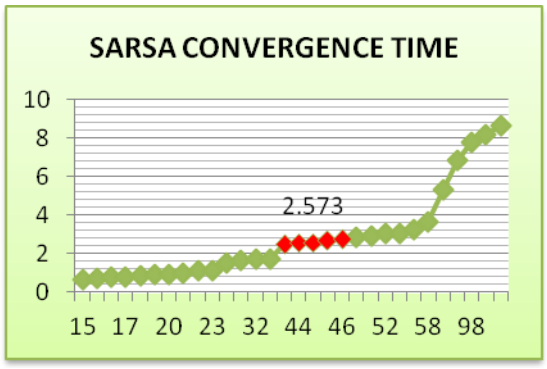
Appendix H3: Monte-Carlo

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	22.326	20	0.893	481
2.	35.101	53	3.721	448
3.	33.910	49	3.323	452
4.	24.434	25	1.222	476
5.	28.870	38	2.194	463
6.	**28.782**	**36**	**2.072**	**465**
7.	31.223	43	2.685	458
8.	35.620	53	3.776	448
9.	17.558	11	0.386	490
10.	31.791	45	2.861	456
11.	33.565	47	3.155	454
12.	20.627	17	0.701	484
13.	34.913	51	3.561	450
14.	23.761	23	1.093	478
15.	20.549	16	0.658	485
16.	33.488	48	3.215	453
17.	27.059	31	1.678	470
18.	32.868	48	3.155	453
19.	30.349	40	2.428	461
20.	26.647	30	1.599	471
21	30.094	39	2.347	462
22.	26.846	29	1.557	472
23.	29.459	37	2.180	464
24	33.704	47	3.168	454
25.	17.747	10	0.355	491
26.	30.298	39	2.363	462
27.	33.930	47	3.189	454
28.	34.234	48	3.286	453
29.	34.776	49	3.408	452
30.	29.916	36	2.154	465



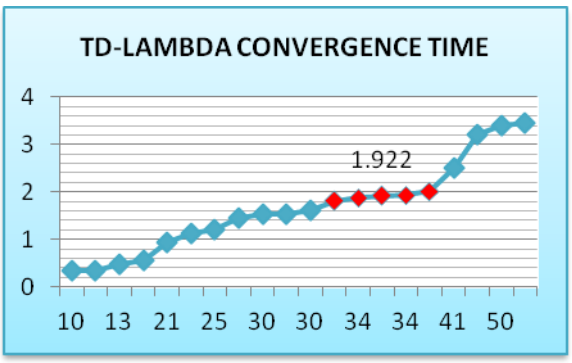
Appendix H4: SARSA

S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	23.775	20	0.951	481
2.	41.637	104	8.660	397
3.	23.423	16	0.750	485
4.	39.788	103	8.196	398
5.	24.869	23	1.144	478
6.	**28.930**	**43**	**2.488**	**458**
7.	29.238	44	2.573	457
8.	29.131	44	2.564	457
9.	30.254	54	3.267	447
10.	31.681	58	3.675	442
11.	39.706	98	7.782	403
12.	30.162	51	3.077	450
13.	34.725	76	5.278	425
14.	25.976	29	1.507	472
15.	38.055	90	6.850	411
16.	22.257	15	0.668	486
17.	23.478	20	0.939	481
18.	23.843	16	0.763	485
19.	25.034	23	1.152	478
20.	26.484	33	1.748	468
21.	23.658	17	0.804	484
22.	25.753	32	1.648	469
23.	27.143	32	1.737	469
24.	24.361	18	0.877	483
25.	29.322	48	2.815	453
26.	23.953	21	1.006	480
27.	30.772	47	2.893	454
28.	29.917	46	2.752	455
29.	29.547	52	3.073	449
30.	29.131	46	2.680	455



Appendix H5: TD-LAMBDA

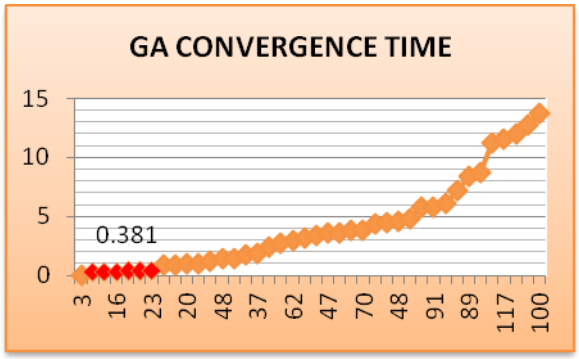
S/N	SIMULATION TIME	CONVERGENCE EPISODE	CONVERGENCE TIME	GOAL
1.	26.778	30	1.607	471
2.	23.453	24	1.126	477
3.	25.676	30	1.541	471
4.	24.368	25	1.218	476
5.	27.708	34	1.884	467
6.	**26.565**	**29**	**1.541**	**472**
7.	28.268	34	1.922	467
8.	27.344	33	1.805	468
9.	28.812	35	2.017	466
10.	16.738	10	0.335	491
11.	18.473	13	0.480	488
12.	22.381	21	0.940	480
13.	18.957	15	0.569	486
14.	30.571	41	2.507	460
15.	33.958	50	3.396	451
16.	26.134	28	1.464	473
17.	28.408	34	1.932	467
18.	34.468	50	3.445	451
19.	16.953	10	0.339	491
20.	33.342	48	3.201	453



Appendix H6: GA

S/N	SIMULATION TIME	CONVERGENCE GENERATION	CONVERGENCE TIME
1.	25.395	20	1.016
2.	25.405	35	1.778
3.	31.509	14	0.882
4.	24.154	62	2.995
5.	18.716	33	1.235
6.	69.515	32	4.449
7.	62.342	35	4.364
8.	24.686	20	0.987
9.	8.064	3	0.048
10.	25.101	76	3.815
11.	30.062	23	1.383
12.	45.864	79	7.247
13.	8.652	22	0.381
14.	9.065	21	0.381
15.	8.278	56	0.927
16.	24.155	50	2.416
17.	27.556	70	3.858
18.	9.188	23	0.423
19.	16.809	143	4.807
20.	28.680	47	2.696
21.	14.318	48	1.375
22.	5.494	30	0.330

23.	31.779	91	5.784
24.	9.838	16	0.315
25.	13.447	116	3.120
26.	38.127	47	3.584
27.	25.613	235	12.038
28.	47.297	48	4.541
29.	47.943	117	11.219
30.	49.314	89	8.778
31.	31.969	91	5.818
32.	10.374	16	0.332
33.	14.458	116	3.354
34.	37.971	47	3.569
35.	25.479	251	12.790
36.	48.061	64	6.151
37.	49.588	117	11.604
38.	46.927	89	8.353
39.	68.707	100	13.741
40.	25.138	37	1.860



Appendix H7: ACO

S/N	CONVERGENCE TIME	CONVERGENCE GENERATION
1.	1.976	26
2.	3.497	128
3.	4.027	158
4.	6.731	304
5.	1.646	23
6.	8.352	405
7.	5.315	229
8.	2.877	92
9.	4.995	215
10.	6.579	309
11.	2.316	63
12.	6.770	318
13.	7.571	368
14.	7.600	370
15.	7.050	342
16.	2.298	52
17.	2.158	53
18.	5.624	248
19.	**6.429**	**303**
20.	3.957	153
21.	8.545	427
22.	9.979	482
23.	4.495	191
24.	3.596	135
25.	7.245	343
26.	7.367	366
27.	1.791	24
28.	5.968	272
29.	2.307	56
30.	6.635	307
31.	7.851	376

32.	1.626	23
33.	8.637	413
34.	2.087	38
35.	5.922	277
36.	4.079	166
37.	6.824	313
38.	3.228	113
39.	5.060	219
40.	3.624	140
41	6.004	268
42	6.191	281
43	9.656	481
44	6.623	308
45	3.908	149
46	3.616	135
47	2.234	55
48	5.589	244
49	3.097	101
50	7.354	320