ABSTRACT

In the capital city of Dhaka, currently, one of the biggest problems is its waste management. Because of over-population, the amount of waste produced every day is so high that it can not be handled properly with the infrastructural aid available now. Transporting the wastes faces a lot of obstacles because of the road and the traffic conditions. As a result, the whole operation is delayed. Furthermore, maintaining a hygienic situation becomes an impossible task. In this work, a modified system will be proposed for dumping wastes in different parts of the city. By using two reinforcement learning techniques (Q-learning, SARSA) imposed in this model, the system will be allowed to find the optimal route for the waste-carrying vehicle so that a faster transportation is ensured so that a suitable state of the environment can be sustained.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Reinforcement Learning is in actuality an area of machine learning. In order that a software agent can be able to take actions in an environment this learning process was devised in which the agent can be set to receive feedbacks from the environment. This will enable it to learn, gradually, and determine the ideal behavior in that environment within a specific context. This will also work towards maximizing the amount of reward while minimizing a large portion of the resources and efforts invested to create scenarios and data to be fed to the agent. As opposed to the standard supervised learning techniques, reinforcement learning differs primarily in minimizing the resource spent in accomplishing the learning process. Furthermore, it maximizes the performance with little use of a human supervisor with expertise on the concerned application domain. Another great side of this is that it can go on for an unspecified amount of time as perfectly functional and yielding output, all the while adapting with time. On the one hand, this nullifies the necessity of the presence of an expert, while on the other hand, with sufficient care in the modeling of the process, several reinforcement learning algorithms have been found prone to converge to the global optimum with the course of time, and thus turns to the ideal behavior that maximizes the reward. Several of the facilities this learning technique provides have encouraged us to implement it in our proposed model. As to the matter of graph algorithms, we have selected the reinforcement learning algorithm over it because the starting states need to be static or predefined in the case of graph algorithms, where our chosen technique has no such necessity like that. Furthermore, reinforcement learning algorithms yield the same results regardless of which state we start from something graph algorithms are unable to do. Reinforcement learning algorithms also provide us with convergence speed which graph algorithms do not. These are the reasons why reinforcement learning algorithm was used in our proposed system of the waste management of Dhaka city.

1.2 Goal

In this thesis, we propose to work with the challenges like proper initialization of the early stages, designing the states, actions, transitions using Markov Decision Process (MDP) and solving the MDP with two popular reinforcement learning techniques namely Q-learning and SARSA(λ). We also want to compare the convergence speed of these two techniques so that we may conclude about one of them to be better.

1.3 Thesis Layout

In the upcoming portion, Chapter 2 contains the various studies we have had to complete in order to get a clear idea on reinforcement learning. Chapter 3 showcases the model that we have proposed in order to solve the wastage dumping problem of Dhaka city, where Chapter 4 includes the detailed method that was used. In Chapter 5, the results obtained from our experiment has been provided. Chapter 6 includes the conclusion we drew from our work and some ideas on possible future work. Finally, a list of the references has been added.

CHAPTER 2

BACKGROUND STUDY

2.1 Reinforcement Learning

For a software agent to take actions in an environment a learning process was devised by implementing a type of machine learning called Reinforcement Learning. This process, in simple terms, enables the agent to receive feedbacks from the environment. Inspired by behaviorist concepts of psychology, Reinforcement Learning (RL) is concerned with the ways a software agent ought to take actions in a particular environment with the aim of maximizing some notion of cumulative reward. Many branches of science see its implementation in various ways. Game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics, and genetic algorithms are some of the sectors that use it regularly. In the operations research and control literature, the field where reinforcement learning methods are studied is called approximate dynamic programming. The problem has seen most of the research on itself in the theory of optimal control, even though most studies are concerned with the existence of optimal solutions and their characterization, but not with the learning or approximation aspects. In the cases of economics and game theory, the methods of reinforcement learning may be used to explain the rise of equilibrium under bounded rationality.

Markov decision process (MDP) is the usual way of formulating the environment in this. Many reinforcement learning algorithms for this context utilize dynamic programming techniques and MDP is a perfect way to do it [1]. The main difference between the classical techniques and reinforcement learning algorithms is that the latter may very easily exclude any knowledge on the MDP and they target large MDPs where exact methods are not to be implemented.

Reinforcement learning differs from standard supervised learning in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Furthermore, on-line performance is the primary component here, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge) [2]. Especially in case of the multi-armed bandit

problem and in finite MDPs, the exploration vs. exploitation trade-off in reinforcement learning has been studied most thoroughly.

2.1.1 Introduction

The basic reinforcement is modeled as a Markov decision process:

1. a set of environment and agent states **S**
2. a set of actions **A** of the agent;
3. $\mathbf{P}_a(s, s') = P_r(\mathbf{s}_{t+1} = s' \mid \mathbf{s}_t = s, a_t = a)$, probability of transition from state **s** to state **s'** under action a.
4. $\mathbf{R}_a(s, s')$, Eimmediate reward after transition from s to s' with action a.
5. Rules that describe what the agent observes.

The rules are often stochastic. In the observation the scalar immediate reward associated with the last transition is typically involved. In many of the related works, it is assumed that the agent observes the current environmental state. In cases of such condition, we talk about full observability, whereas in the opposing case partial observability is discussed. Sometimes restrictions are put upon the set of actions available to the agent. The interaction between a reinforcement learning agent and its environment happens in discrete time steps. At each time **t** the agent receives an observation $\mathbf{a}_t$, which typically includes the reward $\mathbf{r_t}$. It then chooses an action $\boldsymbol{a_t}$ from the set of actions available, which is subsequently sent to the environment. The environment moves to a new state $\mathbf{s}_{t+1}$ and the reward $\mathbf{r}_{t+1}$ associated with the transition $(s_t, a_t, \mathbf{s}_{t+1})$ is determined. The goal of a reinforcement learning agent is to collect as much reward as possible. The agent can choose any action as a function of the history and it can even randomize its action selection.

When we compare the agent's performance to that of an agent acting optimally from the beginning, the difference in performance gives rise to the notion of regret. It is to be noted that in order to act with near-optimal functionality, the agent must reason about the long term consequences of its actions, although the immediate reward associated with this might appear negative.

Thus, reinforcement learning is particularly well-suited to problems that include a long-term versus short-term reward trade-off. Various problems have seen its successful application, including robot

control, elevator scheduling, telecommunications, backgammon and checkers. The pair of components that make reinforcement learning powerful are: (1) the use of samples to optimize performance and (2) the use of function approximation to deal with large environments. Thanks to these two key components, reinforcement learning can be used in large environments in any of the following situations:

- A model of the environment is known, but an analytic solution is not available;
- Only a simulation model of the environment is given;
- The only way to collect information about the environment is by interacting with it.

The first two of these problems could be considered planning problems, while the last one could be considered as a genuine learning problem. However, under reinforcement learning methodology both planning problems would be converted to machine learning problems.

2.2 Exploration

The reinforcement learning problem as described requires clever exploration mechanisms. Randomly selecting actions, without reference to an estimated probability distribution, is known to give rise to very poor performance. What we now understand the case of (small) finite Markov decision processes is comparatively better than that of any earlier success. However, due to the lack of algorithms that would probably scale well with the number of states, in practice, simple exploration methods are often used. One such method is $\in$ -greedy, when the agent chooses the action that it believes has the best long-term effect with probability $1-\in$, and it chooses an action uniformly at random. Here, $0 < \in < 1$ is a tuning parameter, which is sometimes changed, either according to a fixed schedule, or adaptively based on some heuristics [3].

2.3 Algorithms for Control Learning

Even if the issue of exploration is disregarded and even if the state was observable, the problem remains to find out which actions are good based on past experience

2.3.1 Criterion of optimality

The action selection of the agent is modeled as a map called policy:

$$\boldsymbol{\pi : S * A \rightarrow [0, 1]}$$
$$\boldsymbol{\pi(\alpha|s) = P(a_t = a|s_t = s)}$$

The policy map gives the probability of taking action **a** when in state **s** [4],

Value function V_π is defined as the expected return starting with state $\boldsymbol{s}$ and policy. Value function estimates how good it is to be in a given state.

$$V_\pi(s) = E[\, R \,] = E[\, \sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \,]$$

Where the random variable R denotes the return and is defined as the sum of discounted rewards.

$$R = \sum_{t=0}^{\infty} \gamma^t r_t$$

The problem then is to specify an algorithm that can be used to find a policy with maximum expected return. From the theory of MDPs it is known that, without loss of generality, the search can be restricted to the set of the so-called *stationary* policies. A policy is called stationary if the action-distribution returned by it depends only on the last state visited. In fact, the search can be further restricted to *deterministic* stationary policies. A deterministic stationary policy is one which deterministically selects actions based on the current state [5]. Since any such policy can be identified with a mapping from the set of states to the set of actions, these policies can be identified with such mappings with no loss of generality.

2.3.2 Value function approach

Value function approaches attempt to find a policy that maximizes the return by maintaining a set of estimates of expected returns for some policy .These methods rely on the theory of MDPs, where

optimality is defined in a sense which is stronger than the above one: A policy is called optimal if it achieves the best expected return from *any* initial state. Again, one can always find an optimal policy amongst stationary policies.

To define optimality in a formal manner, define the value of a policy **π** by

$$V^{\pi}(s) = E\ [\ R|s, \pi]$$

Where **R** stands for the random return associated with following π from the initial state **s** . Define V^{*}(s) as the maximum possible value of V^{π} (s), where is allowed to change [6].

Although state-values suffice to define optimality, it will prove to be useful to define action-values. Given a state s, an action a and a policy π , the action-value of the pair (s, a) under π is defined by

$$Q^{\pi}(s, a) = E\ [\ R|s, a, \pi\ ]$$

Where, now, **R** stands for the random return associated with first taking action **a** in state **s** and following π , thereafter.

It is well-known from the theory of MDPs that if someone gives us Q for an optimal policy, we can always choose optimal actions by simply choosing the action with the highest value at each state. The action-value function of such an optimal policy is called the optimal action-value *function* and is denoted by Q^{*} . In summary, the knowledge of the optimal action-value function *alone* suffices to know how to act optimally [7].

Assuming full knowledge of the MDP, there are two basic approaches to compute the optimal action-value function, value iteration and policy iteration. Both algorithms compute a sequence of functions Q_k (k=0, 1, 2, 3…) which converge to Q^{*}. Computing these functions involves computing expectations over the whole state-space, which is impractical for all but the smallest MDPs, never mind the case when the MDP is unknown. In reinforcement learning methods the expectations are approximated by averaging over samples and one uses function approximation techniques to cope with the need to represent value functions over large state-action spaces.

2.3.3 Monte Carlo Methods

Algorithms that mimic policy iteration can work with the simplest Monte Carlo methods. Policy iteration consists of two steps: (1) policy evaluation and (2) policy improvement.The Monte Carlo methods are used in the policy evaluation step. In this step, given a stationary, deterministic policy π, the goal is to compute the function values $Q^{\pi}(s, a)$ for all state-action pairs (s, a) . Assume (for simplicity) that the MDP is finite and in fact a table representing the action-values fits into the memory. Further, assume that the problem is episodic and after each episode a new one starts from some random initial state. Then, the estimate of the value of a given state-action pair (s, a) can be computed by simply averaging the sampled returns which originated from (s, a) over time. Given enough time, this procedure can thus construct a precise estimate Q of the action-value function Q^{π}. This finishes the description of the policy evaluation step [8].

In the policy improvement step, as it is done in the standard policy iteration algorithm, the next policy is obtained by computing a greedy policy with respect to Q: Given a state s, this new policy returns an action that maximizes $Q(s, a)$. In practice one often avoids computing and storing the new policy, but uses lazy evaluation to defer the computation of the maximizing actions to when they are actually needed [9] . A few problems with this procedure are as follows:

- The procedure may waste too much time on evaluating a suboptimal policy;
- It uses samples inefficiently in that a long trajectory is used to improve the estimate only of the single state-action pair that started the trajectory;
- When the returns along the trajectories have high variance, convergence will be slow;
- It works in episodic problems only;
- It works in small, finite MDPs only.

2.3.4 Temporal difference methods

The first issue is easily corrected by allowing the procedure to change the policy (at all, or at some states) before the values settles. However good may this sound, there are chaces of it being

problematic, as this might prevent convergence. Still, most current algorithms implement this idea, giving rise to the class of generalized policy iteration algorithm. It is to be noted that many actor critic methods belong to this category. The second issue can be corrected within the algorithm by allowing trajectories to contribute to any state-action pair in them. Batch methods, a prime example of which is the least-squares temporal difference method [10], may use the information in the samples better, whereas incremental methods are the only choice when batch methods become infeasible due to their high computational or memory complexity. In addition, there exist methods that try to unify the advantages of the two approaches. Methods based on temporal differences also overcome the second but last issue. In order to address the last issue mentioned in the previous section, function approximation methods are used. In linear function approximation one starts with a mapping $\emptyset$ that assigns a finite-dimensional vector to each state-action pair. Then, the action values of a state-action pair (s, a) are obtained by linearly combining the components $\emptyset(\mathrm{s}, \mathrm{a})$ of with some weights θ

$$Q(s,a) = \sum_{i=1}^{d} \theta_i \emptyset_i (s,a)$$

The algorithms then adjust the weights, instead of adjusting the values associated with the individual state-action pairs. However, linear function approximation is not the only choice. More recently, methods based on ideas from nonparametric statistics have been explored.

So far, the discussion was restricted to how policy iteration can be used as a basis of the designing reinforcement learning algorithms. Equally importantly, value iteration can also be used as a starting point, giving rise to the Q-Learning algorithm and its many variants [11].

The problem with methods that use action-values is that,. For them, it is necessary to have highly precise estimates of the competing action values, which can be hard to obtain when the returns are noisy. Though this problem is mitigated to some extent by temporal difference methods and if one uses the so-called compatible function approximation method, more works remains to be done to increase generality and efficiency. Another problem specific to temporal difference methods comes from their reliance on the recursive Bellman equation. Most temporal difference methods have a so-called λ parameter ($(0 \leq \lambda \leq 1)$ that allows one to continuously interpolate between Monte-Carlo methods and the basic temporal difference methods, which can thus be effective in palliating this issue [12].

CHAPTER 3

3. Proposed System Model

3.1. Introduction

This section consists of the detailed description of the model we are proposing which contains several phases (see Figure 1).

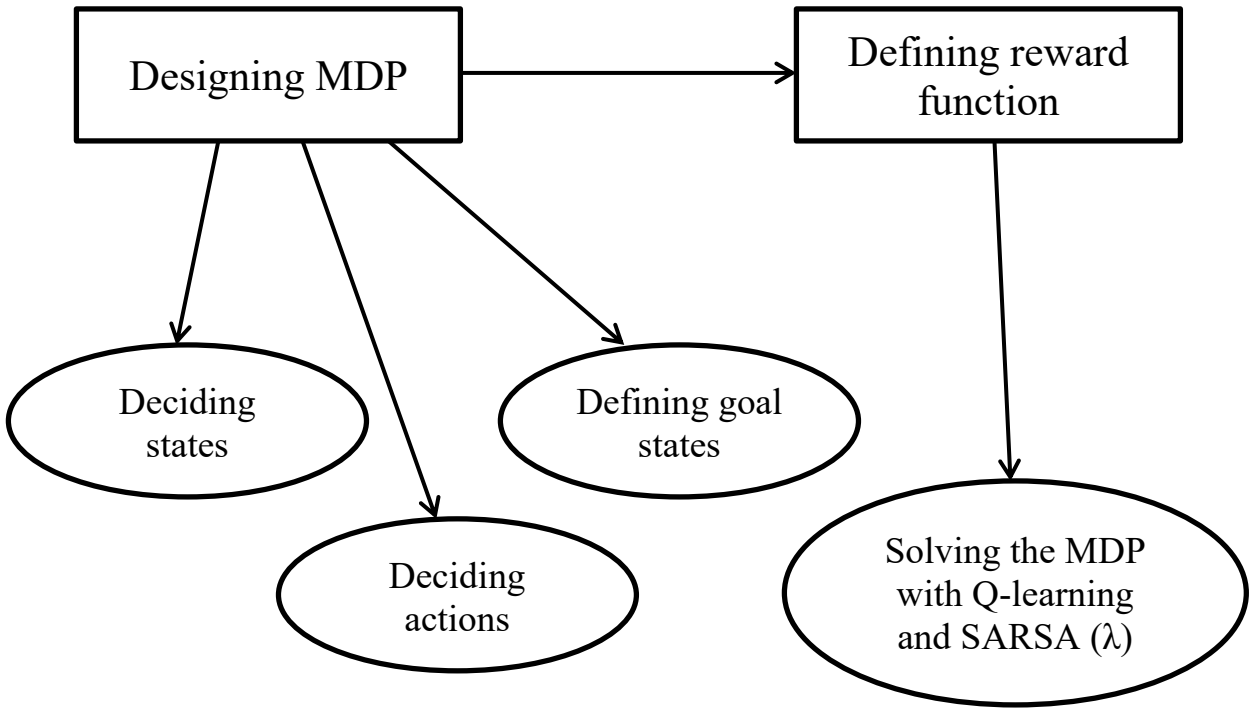

Figure 1: Workflow of the proposed system model

3.2. Markov Decision Process

Since our proposed model deals with the route selection for the waste management in Dhaka city, we are to consider several places of the city as states in this system. In order for this system to work, it is essential to calculate the distances among them and the amounts of time spent to cover those distances. So, to find the optimal route, we intend to use the Markov Decision Process (MDP) model and the computation is to be done by reinforcement learning. An MDP has a decision agent to repeatedly and continuously observe the current state of the system [19]. After the close observation it takes a decision that is allowed to be taken in that state and then observes a transition to a new state. A reward influences the decisions of the agent.

An MDP model contains:

1. A set of possible states S

2. A set of possible actions A

3. A real valued reward function R(s, a)

4. A description T of each action's effects in each state.

5. Stochastic actions:

T: S ×A → Prob(S), for each state and action we specify a new Probability distribution over next states. Representation of the distribution is P(s ′ |s, a).

To solve our optimal route selection problem, we have designed two MDPs (Markov Decision Process) one of which is for Dhaka North City Corporation (DNCC) (see Figure 2) and the other of which is for Dhaka South City Corporation (DSCC) (see Figure 3). The MDPs will contain several states and actions. The MDPs are necessary for generating episodic decision making policies for our problem. In our work, we propose to use Markov Decision Process (MDP) model. An MDP involves a decision agent that repeatedly observes the current states of the controlled system, takes a decision among the ones allowed in that state and then observes a transition to a new state s' and a reward r that will drive its decisions [13]. The MDP that we will be using for our work is as follows:

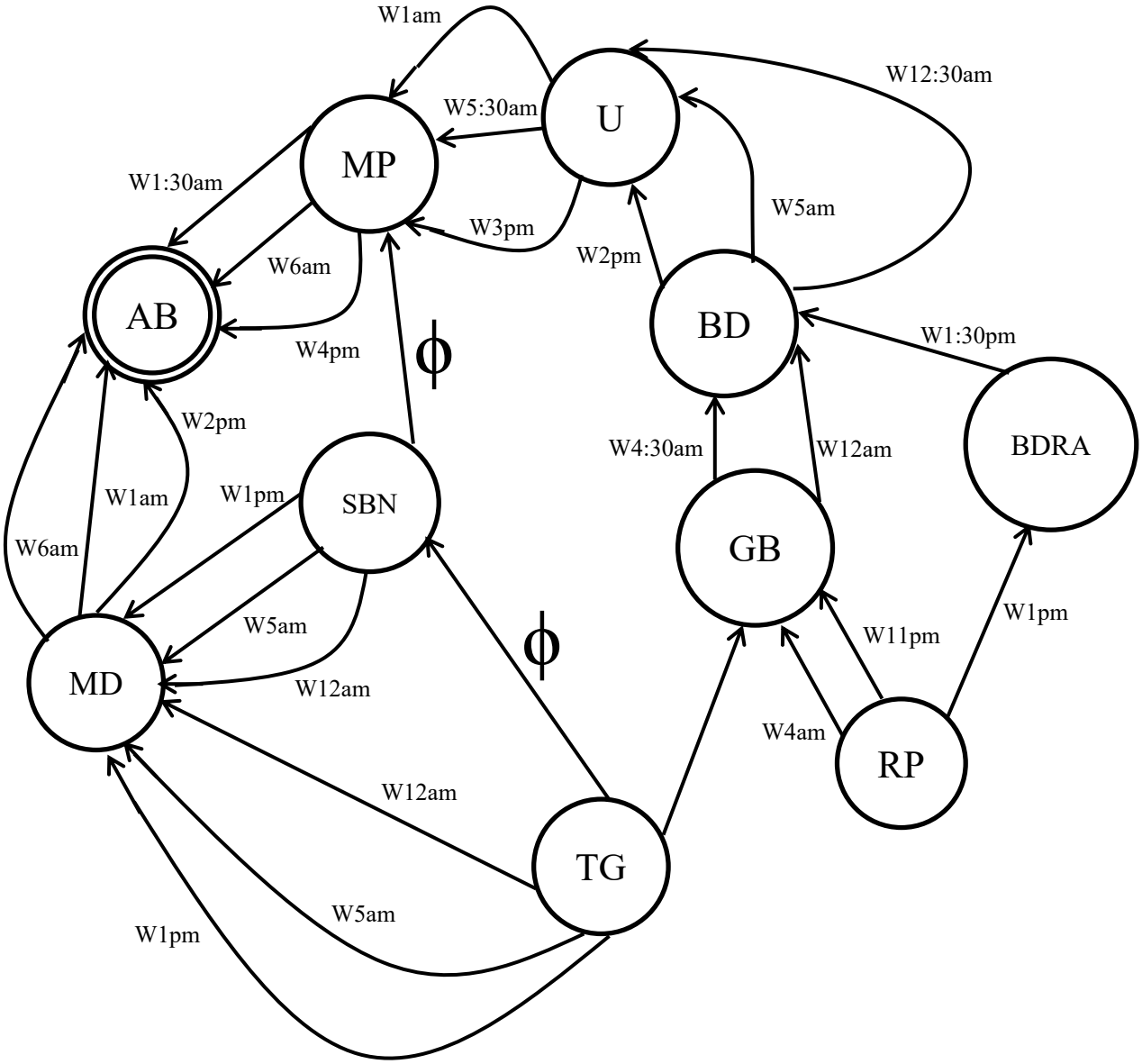

Figure 2: MDP of Dhaka North City Corporation (DNCC)

M= {S, A, T, R, β} where:

S= {RP, GB, BD, BDRA, U, MP, TG, MD, SBN, AB}

For our work, we have divided Dhaka North City Corporation (DNCC) into a number of zones which are represented in the diagram above as the states of the system.

Here,

RP represents Rampura, Banashree and the surrounding areas.
GB represents Gulshan, Banani and the surrounding areas.
BD represents Baridhara and the surrounding areas.
BDRA represents the Bashundhara Residential Area.
U represents Uttara, Khilkhet and the surrounding areas.
MP represents the Mirpur area.
TG represents Tejgaon and the surrounding areas.
MD represents Mohammadpur, Kalyanpur and the surrounding areas.
SBN represents the Sher-E-Bangla Nagar areas.
AB represents Aminbazaar, which is the dumping station.

A represents the action set we have used for our work, such as **W4am**, **W11pm**, **W1pm**, etc. **W4am**, **W11pm**, **W1pm**, here, mean that the truck carrying the wastes must arrive from the **RP** zone to the **GB** zone and the **BDRA** zone within 1 PM and 4 AM, 11 AM respectively. Similarly, the rest of our actions are:

{W1:30pm, W2pm, W3pm, W4pm, W12am, W12:30am, W1am, W1:30am, W4:30am, W5am, W5:30am, W6am}

In the MDP, the symbol ϕ was used, which represents a route that will not be used.

T is the probability distribution of going to a state "s′" from "s" by taking any random action "a".

R is the cost function that expresses the reward if action "a" is taken at state *s*.

"β" is the discount factor, $0 < \beta < 1$.

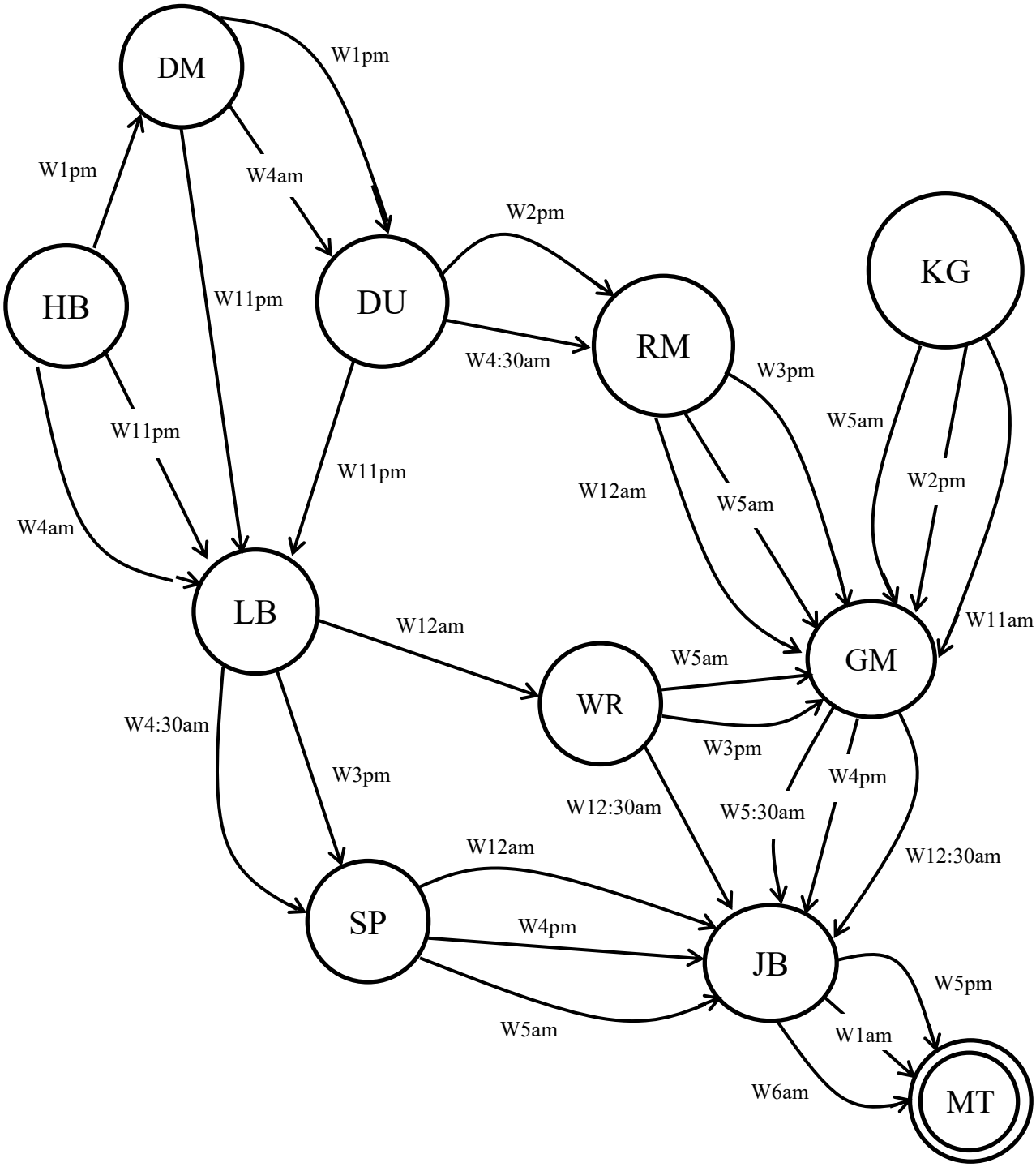

Figure 3: MDP of Dhaka South City Corporation (DSCC)

S= {HB, DM, LB, DU, SP, WR, RM, JB, GM, KG, MT}

For our work, we have divided Dhaka South City Corporation (DSCC) into a number of zones which are represented in the diagram above as the states of the system.

Here,

HB represents Hazaribagh area that produces an immense amount of waste from tanneries
DM represents Dhanmondi, Lalmatia and the surrounding areas.
LB represents Lalbagh, Chawk Bazar and the surrounding areas.
DU represents the entire Dhaka University area.
SP represents Sutrapur, Narinda, Shamibag and the surrounding areas.
WR represents Wari, Tikatuli and the surrounding areas.
RM represents Ramna, Shahbagh, Elephant Road and the surrounding areas.
GM represents Gulistan, Motijheel, Paltan, Kakrail and the surrounding areas.
KG represents Khilgaon, Basabo, Goran, Mathartek, Manda and the surrounding areas.
JB represents Jatrabari, Gandaria, Doniya and the surrounding areas.
MT represents Matuail, which is the dumping station.

A represents the action set we have used for our work, such as **W4am**, **W4:30am**, **W5am**, etc. **W4am**, **W4:30am**, **W5am**, here, mean that the truck carrying the wastes must arrive from the **HB** zone to the **LB** zone and the **SP** zone within 4 AM and 4:30 AM, 5 AM respectively. Similarly, the rest of our actions are:

{W4am, W1am, W11pm, W4:30am, W1pm, W3pm, W12am, W4:30am, W5am, W4pm, W12:30am, W5:30am, W6am, W2pm, W11am}

CHAPTER 4

4. PROPOSED METHOD

4.1 Q-learning

The reinforcement learning technique we used here is q-learning. Q-learning is a model free reinforcement learning technique. It works by learning an action value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. Our Q – learning algorithm is [14]:

A. Q-learning

1. $(\forall s \in S)(\forall a \in A(s))$;
2. **initialize** Q(s , a)
3. s := the initial observed state
4. **loop**
5. Choose $a \in A(s)$ according to a policy derived from Q
6. Take action a and observe next state s' and reward r
7. $Q[s , a] := Q[s , a] + \alpha(R[s,a] + * \max_a Q[s', a'] - Q[s, a])$
8. $s := s'$
9. **end loop**
10. return $\pi(s) = \text{argmax}_a Q(s , a)$

Here, "α" is the learning rate. It determines to how much the old information will be wiped out by the newer one. Value of α being "0" will make the agent not to learn anything and on the contrary value of α being "1" would make it consider only the recent most information. In deterministic environments the value of α can be set to 1 and that is optimal. But our environment is stochastic and it is quite tough to determine the exact value. "" is the discount factor. It determines how important the future rewards can be. A value of "0" will make the agent short sighted and the agent will only consider the current rewards.

4.2 SARSA(λ)

State-Action-Reward-State-Action (SARSA) is another reinforcement algorithm to solve MDP. The name simply reflects that the function that updates the Q value depends on the current state of "s", the action "a", the reward "r" that an agent gets by choosing the action a and the next state "s′". When eligibility traces are added to SARSA algorithm, the algorithm is called SARSA (λ) algorithm [15]. Our SARSA (λ) algorithm is given below [14]:

1. **Initialize** Q(s, a) arbitrarily
2. **Repeat** (for each episode):
3. **Initialize** s
4. Choose a from s using policy derived from Q
5. **Repeat** (for each episode):
6. Take action a, observe r, s′
7. Choose a′ from s′ using policy derived from Q
8. **δ** = r+ Q[s′ , a′] - Q[s, a]
9. e(s, a) = e(s, a)+1
10. **For** all (s , a):
11. Q[s , a] = Q[s , a] + α **δ** e(s, a)
12. e(s, a) = λe(s, a)
13. s=s′ ; a=a′
14. **until** s is terminal

Eligibility trace is a very important term in SARSA (λ) algorithm. There are two ways to view eligibility traces. The more theoretical view, which we emphasize here, is that they are a bridge from TD to Monte Carlo methods. When TD methods are augmented with eligibility traces, they produce a family of methods spanning a spectrum that has Monte Carlo methods at one end and one-step TD methods at the other. In between are intermediate methods that are often better than either extreme method [8]. In this sense eligibility traces unify TD and Monte Carlo methods in a valuable and revealing way.

The other way to view eligibility traces is more mechanistic. From this perspective, an eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a TD error occurs, only the eligible states or actions are assigned credit or blame for the error [17]. Thus, eligibility traces Help Bridge the gap between events and training information. Like TD methods themselves, eligibility traces are a basic mechanism for temporal credit assignment.

4.3 Reward Function

To experiment with the Q-learning and SARSA (λ), we have defined the reward function that has been used is as follows:

R=β (Cost) + (1-β) (Penalty)

Where,

Cost = $V_n * (F_c + M_c + L_c)$ ..... (1)

Penalty = $P_c * (1 + (P_d - P_{sla})/ P_{sla})$ .....(2)

In equation (1),

V_n = Number of vehicles

F_c = the cost of fuel

M_c = cost of maintenance

L_c = cost of labors

In equation (2),

P_c = penalty for the violation of SLA

P_d = the performance displayed by the system randomly

P_{sla} = target performance

Lastly, β is the balancing factor

CHAPTER 5

5. EXPERIMENTAL RESULTS

5.1 Variant Beta (β)

We varied the β in accordance with the cost and penalty we acquire in different training episodes and plot them in a graph while implying q-learning. We also did the same in case of SARSA (λ) [16]. The following table shows us the average (random 10 episodes) of the cost and penalties for different parameters of beta for Q-learning (see Figure 4, Table 1).

Value of β	Cost	Penalty
0.10	35.27	8.4
0.25	73.19	7.26
0.50	77.35	9.08
0.75	82.63	8.13
0.90	91.85	8.39

Table1. Cost and Penalty for variant beta (Q)

The graph below shows which value of β balances the cost and Penalty:

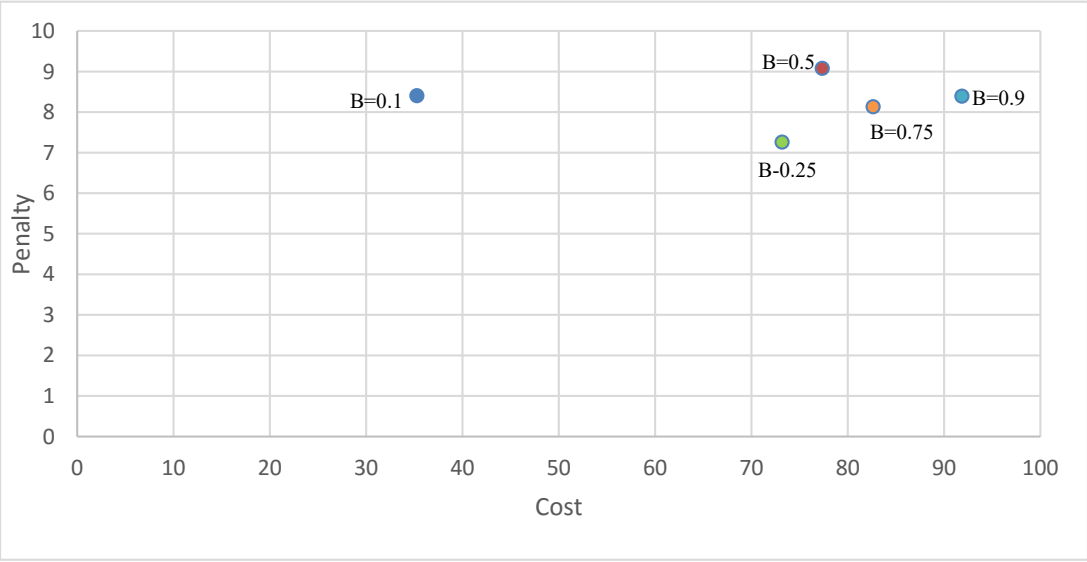

Figure: 4 Cost Vs. Penalty Graph for beta in Q -learning

Again, the following table shows us the average (random 10 episodes) of the cost and Penalties for different parameters of beta for SARSA (λ) (see Figure 5, Table 2).

Value of β	Cost	Penalty
0.10	2.85	7.21
0.25	3.67	6.04
0.50	5.23	8.07
0.75	7.16	6.39
0.90	9.25	6.18

Table 2. Cost and Penalty for variant beta (SARSA)

The graph below shows which value of β balances the cost and Penalty for SARSA (λ):

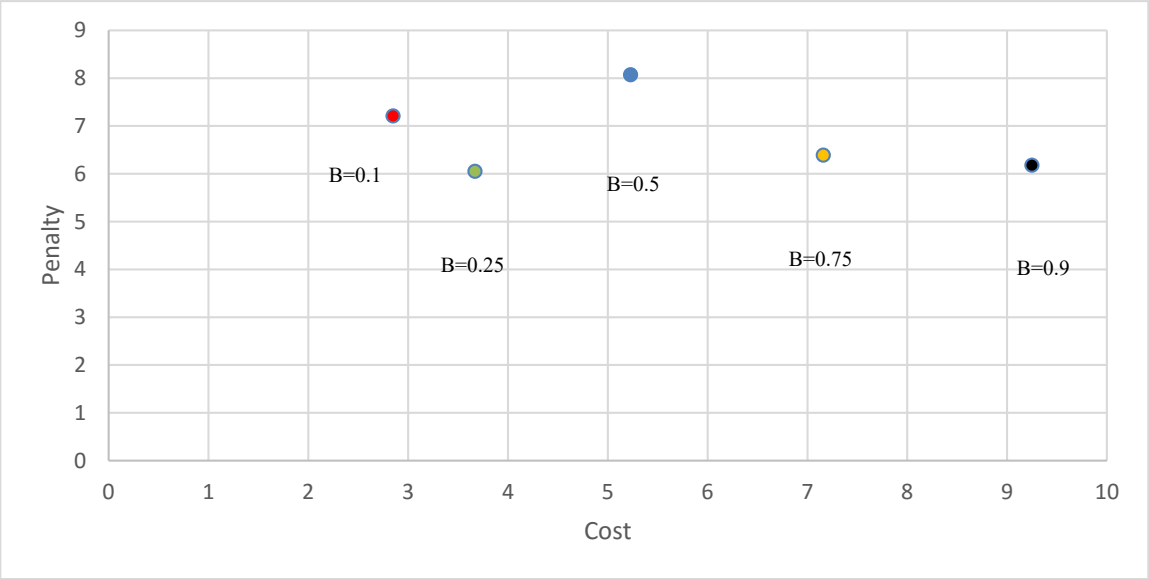

Figure 5: Cost Vs. Penalty Graph for beta in SARSA- λ

To compare the beta values of these two reinforcement learning techniques, we merged the graphs stated above and observed the versatile values of beta. The graph below shows us the comparison of the beta values for both of the learning techniques (see Figure 6):

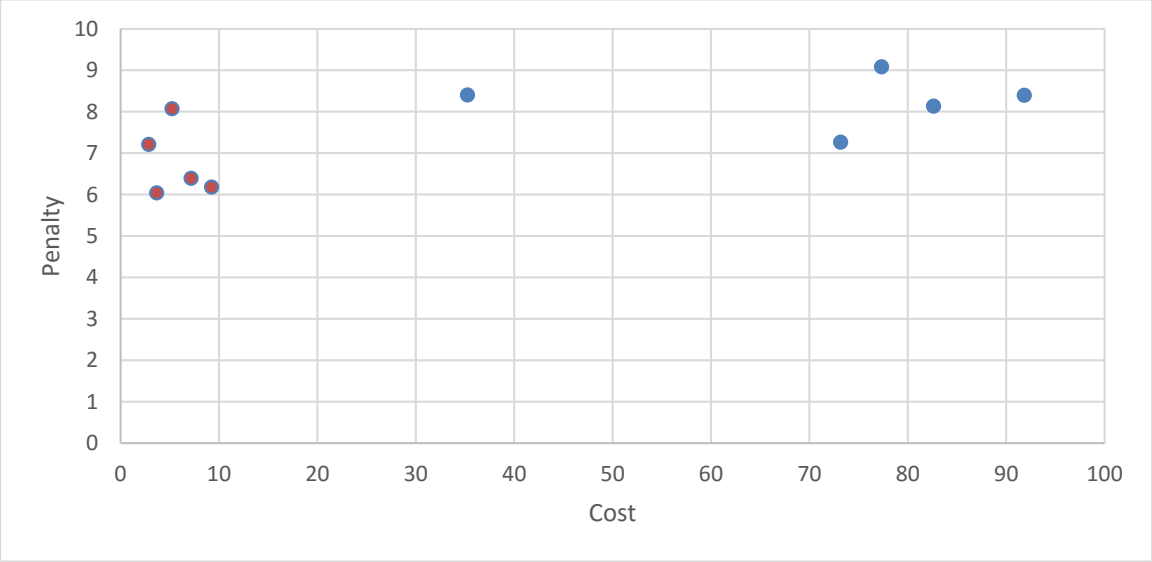

Figure 6: Cost Vs. Penalty Graph for beta in Q and SARSA(λ)

Here, in the graph, the purple dots represent the cost versus penalty results of the effects of Q-learning algorithm, and the blue dots represent the cost versus penalty results of the effects of SARSA algorithm.

5.2 Variant Lambda (λ)

For SARSA (λ) algorithm, we also varied the values of lambda to see which value of lambda best balances the reverse condition between cost and penalty. The values of lambda taken on account are 0.1, 0.25, 0.5, 0.75, and 0.9. It gave us the following result (see Figure 7, Table3):

Value of λ	Cost	Penalty
0.1	11.87	7.13
0.25	9.25	6.18
0.5	8.31	6.03
0.75	10.63	8.29
0.9	7.06	10.67

Table 3. Cost and Penalty for variant lambda (SARSA)

The values gave us the following result:

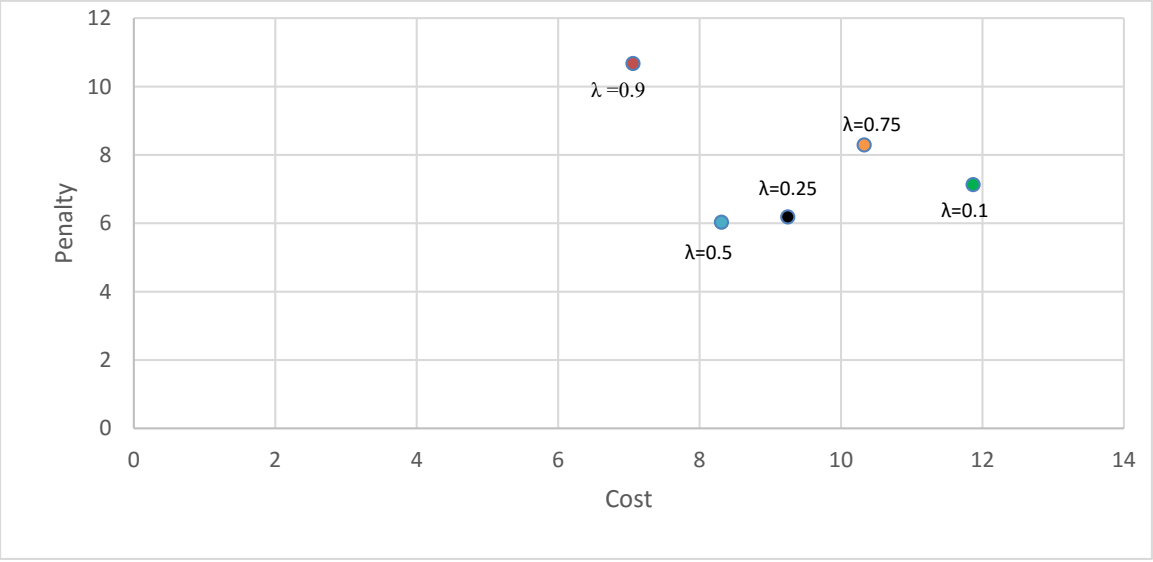

Figure 7: Cost Vs. Penalty Graph for λ in SARSA (λ)

5.3 Variant Alpha (α)

To decide up to what extent the newly acquired information will override the old information, learning rate was varied throughout the experiment while applying Q- learning. The values of alpha that we varied were 0.1, 0.25, 0.5, 0.75 and 0.9. These values generated variant results with rewards. Alpha was chosen as 0.1 because this is the only value of alpha in which the convergence took place. The following graph represents different values of alpha generating chunks of reward (see Figure 8):

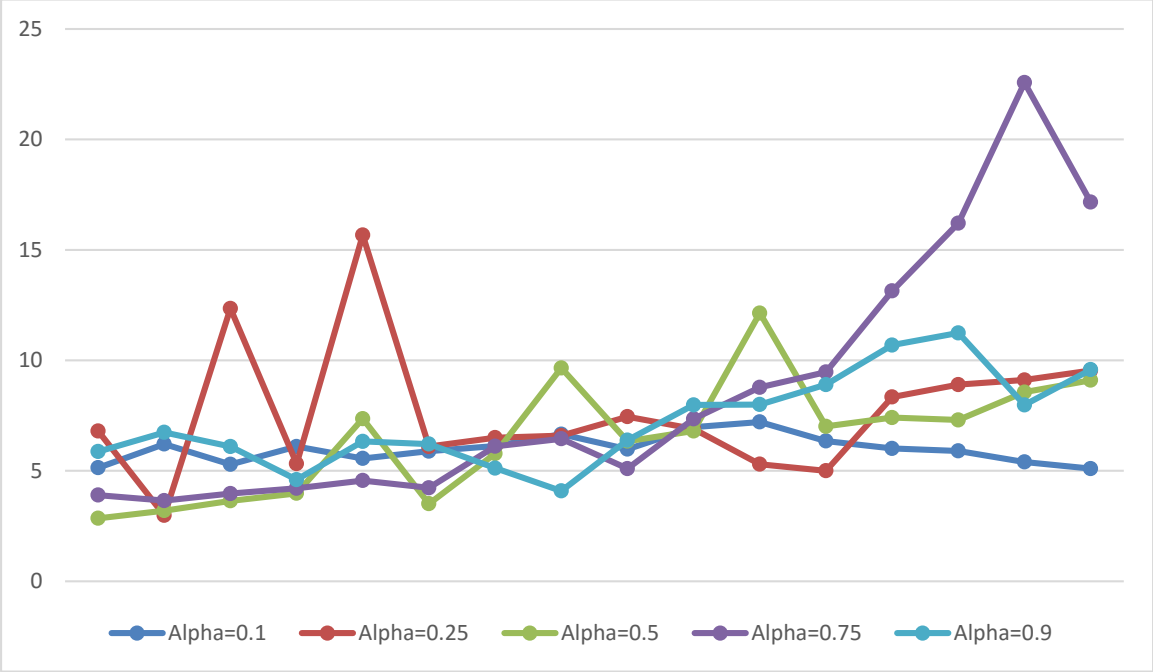

Figure 8: Different values of alpha producing chunks of reward (Q-Learning)

Learning rate was varied throughout the experiment while applying SARSA-lambda too. The values of alpha that we varied were 0.1, 0.25, 0.5, 0.75 and 0.9. These values generated variant results with rewards. Alpha was chosen as 0.1 because this is the only value of alpha in which the convergence took place. The following graph represents different values of alpha generating chunks of reward (see Figure 9):

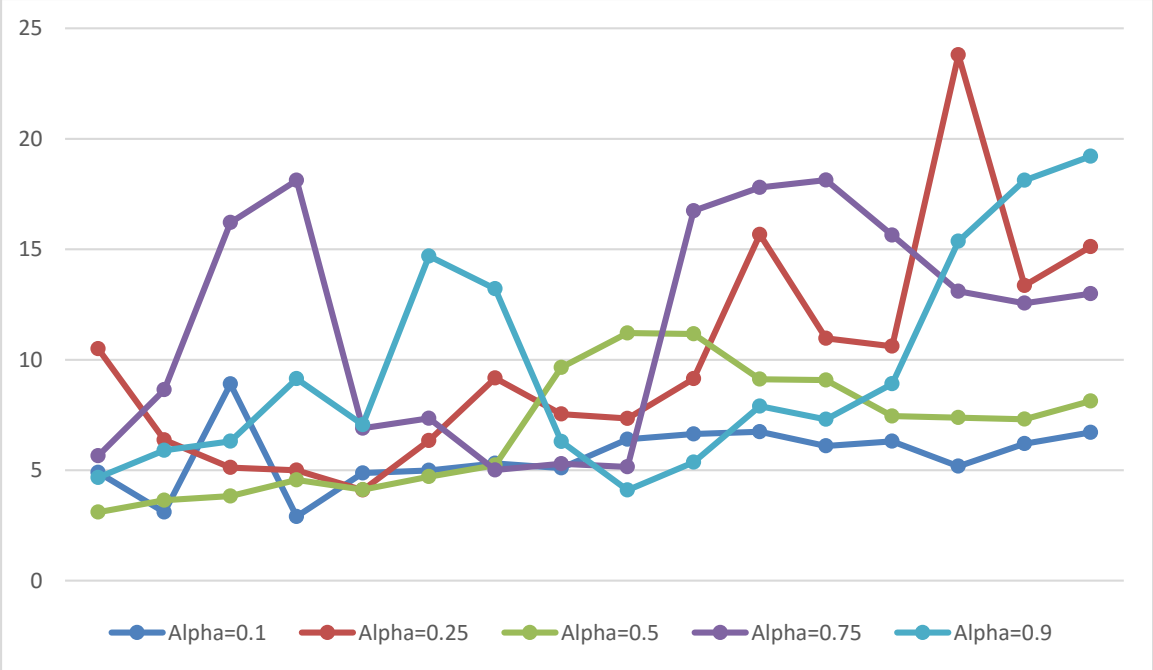

Figure 9: Different values of alpha producing chunks of reward (SARSA)

5.4 Optimal Route after implying the Algorithms

For the best implementation, we have divided the 24-hour day in 3 shifts in which the dumping trucks are to carry the wastes. For a clearer understanding, this distribution is described below.

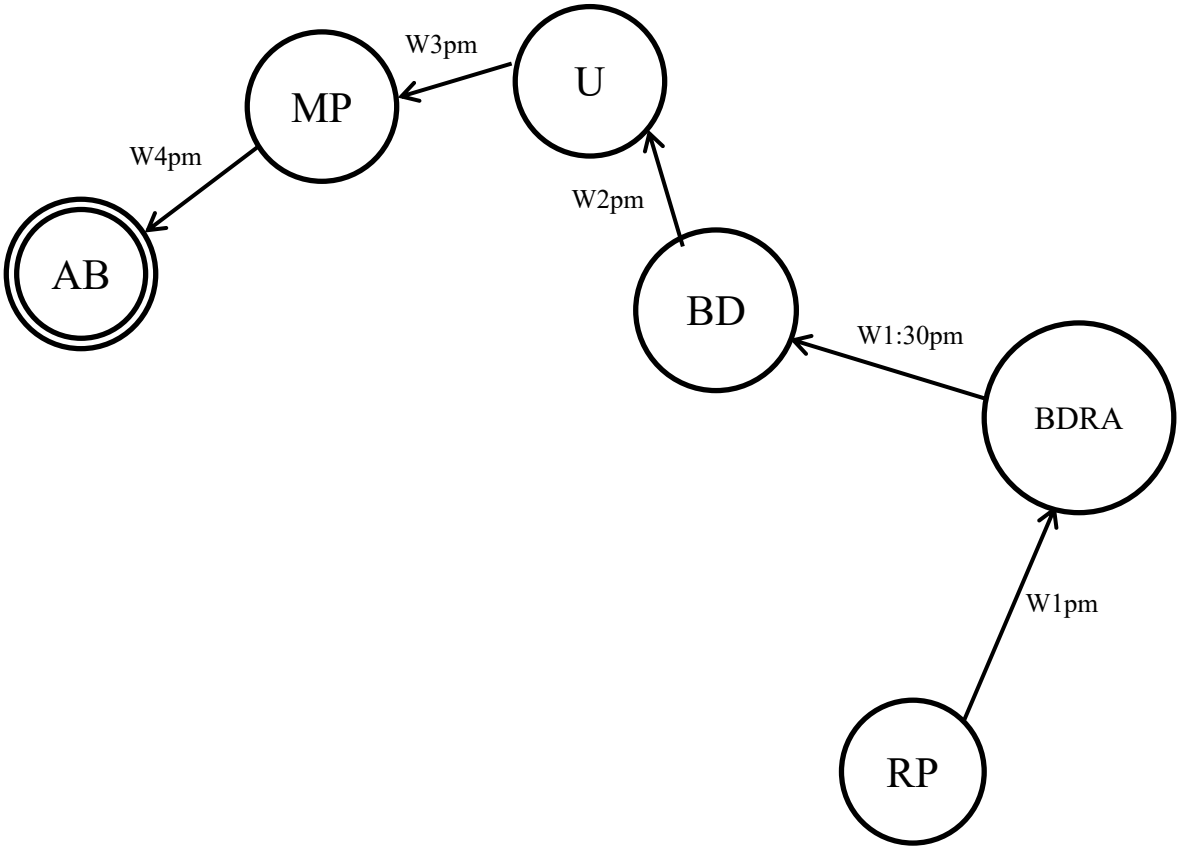

Figure 10: Route from Rampura zone to Aminbazar dumping station (day shift)

Figure 10 shows that, in the day shift, the dumping truck starts from Rampura zone (RP) at 1 PM and travels this route through Bashundhara Residential Area zone (BDRA), Baridhara zone (BD), Uttara zone (U), Mirpur zone (MP), and at the end reaches and dumps the waste at Aminbazar dumping station (AB) within 4 PM.

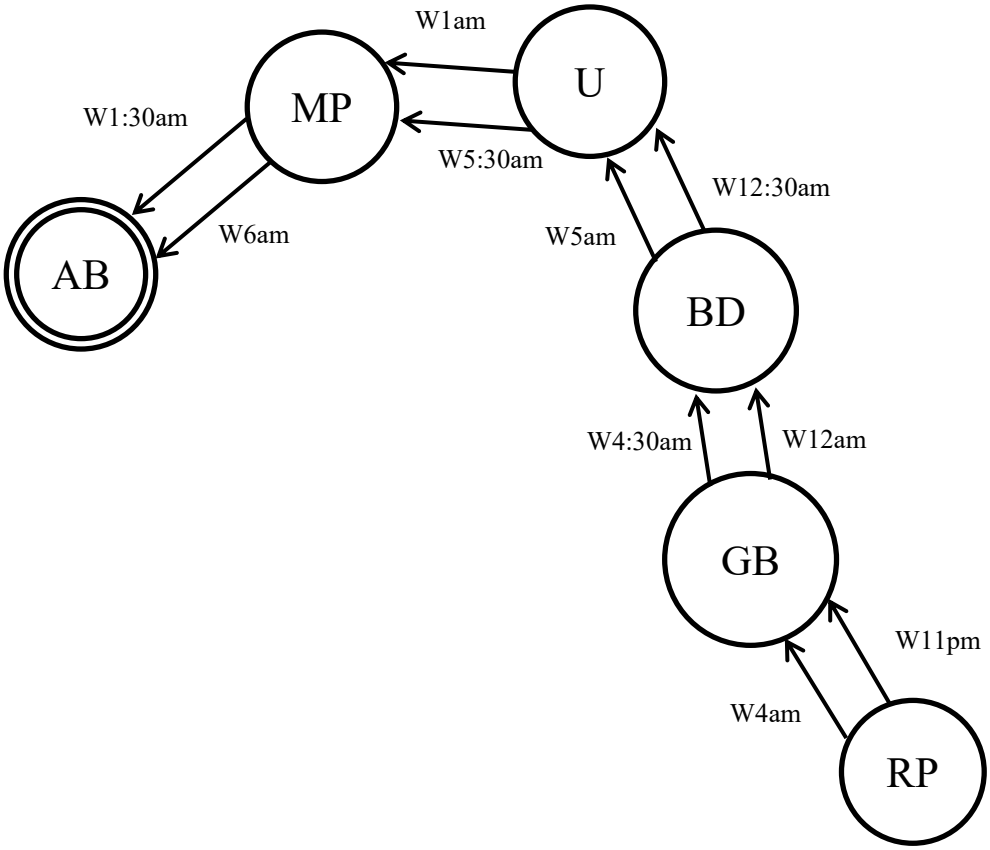

Figure 11: Route from Rampura zone to Aminbazar dumping station (night and dawn shifts)

Figure 11 shows that, in the night shift, the dumping truck starts from Rampura zone (RP) at 11 PM and travels this route through Gulshan-Banani zone (GB), Baridhara zone (BD), Uttara zone (U), Mirpur zone (MP), and at the end reaches and dumps the waste at Aminbazar dumping station (AB) within 1:30 AM. Following the same route. In the dawn shift, the dumping truck starts from Rampura zone (RP) at 4 AM and reaches and dumps the waste at the Aminbazar dumping station (AB) within 6 AM.

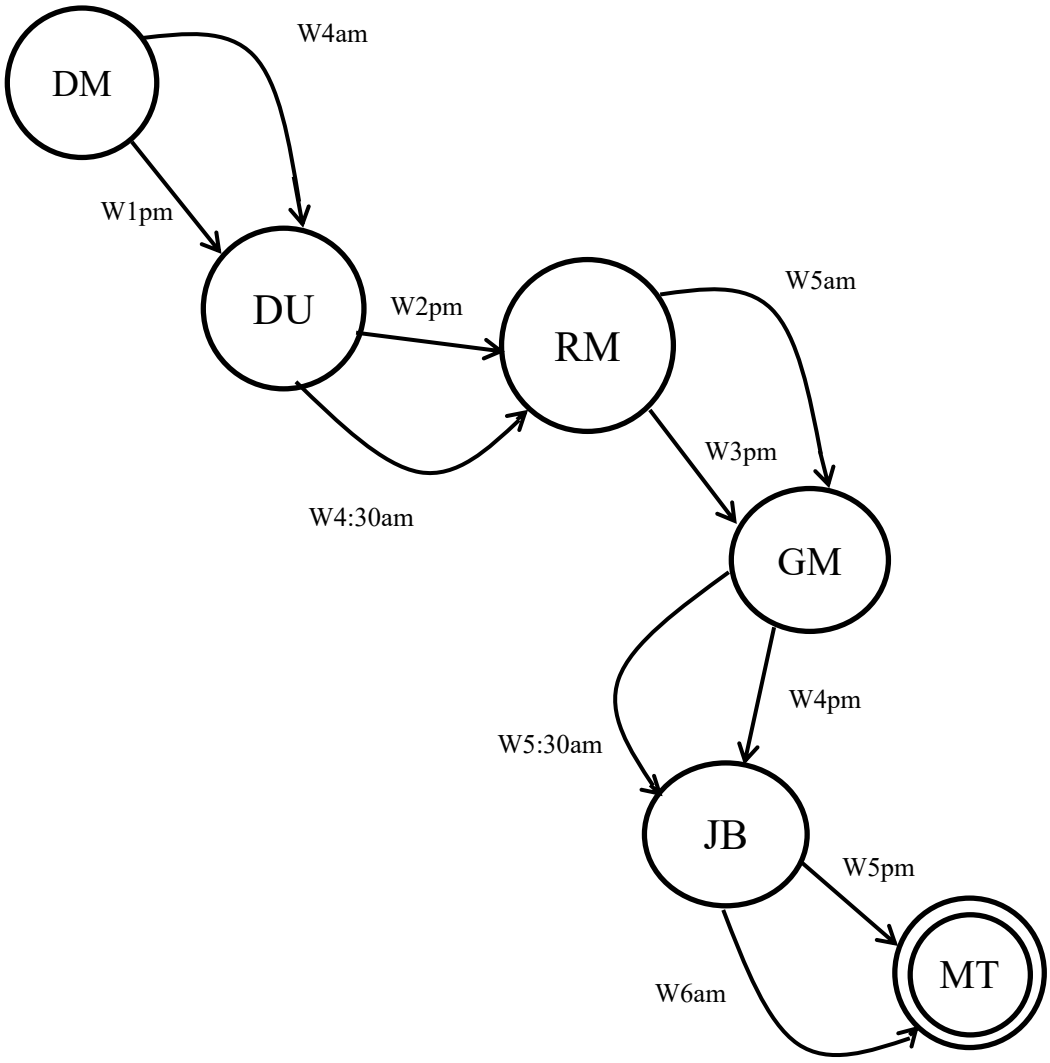

Figure 12: Route from Dhanmondi zone to Matuail dumping station (dawn and day shifts)

Figure 12 shows that, in the dawn shift, the dumping truck starts from Dhanmondi zone (DM) at 4 AM and travels this route through Dhaka University area zone (DU), Romna zone (RM), Gulistan-Motijheel zone (GM), Jatrabari zone (JB) and at the end reaches and dumps the waste at Matuail dumping station (MT) within 1:30 AM. In the day shift, the truck starts from Dhanmondi zone (DM) at 1 PM and travels the same route and at the end reaches and dumps the waste at Matuail dumping station (MT) within 5 PM.

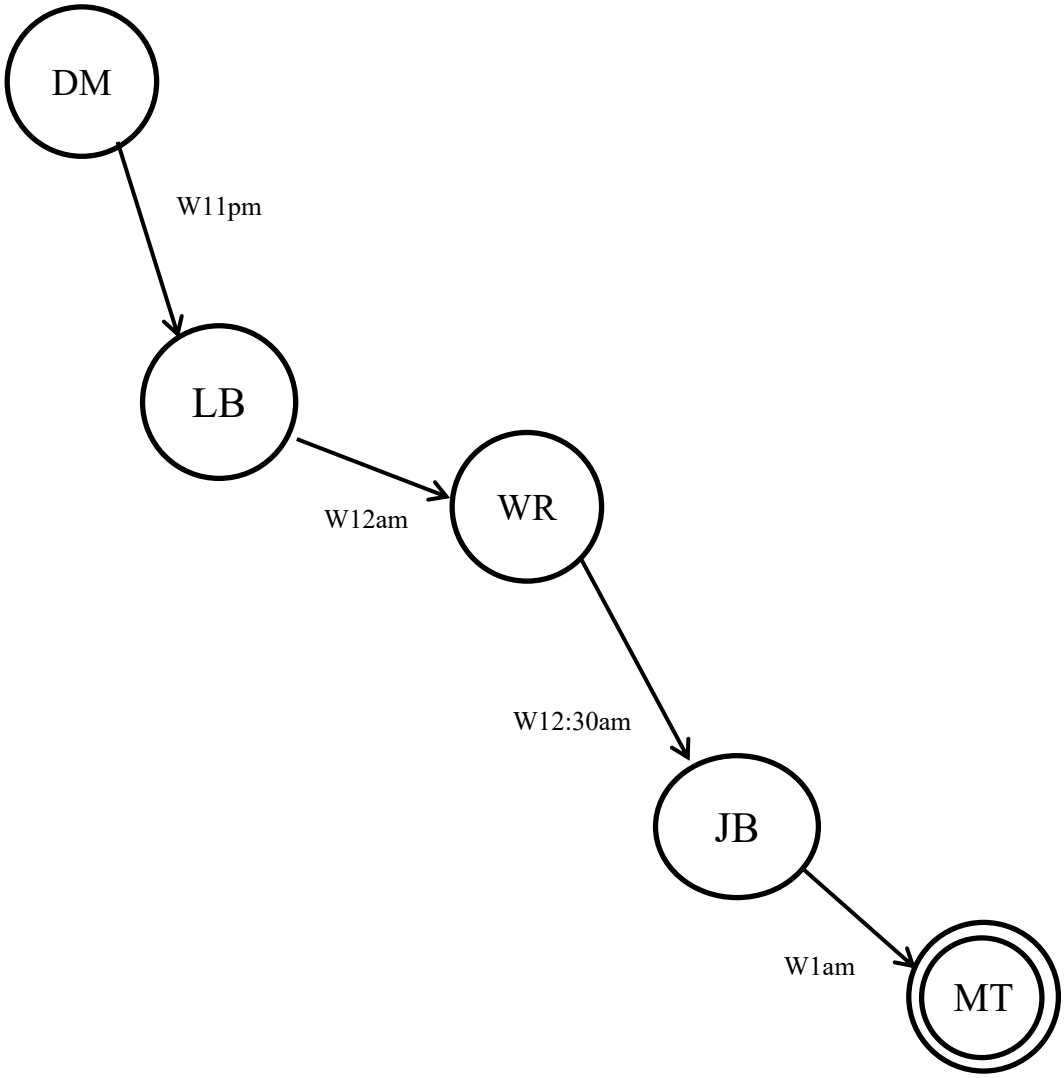

Figure 13: Route from Dhanmondi zone to Matuail dumping station (night shift)

Figure 13 shows that, in the night shift, the dumping truck starts from Dhanmondi zone (DM) at 11 PM and travels this route through Lalbagh zone (LB), Wari zone (WR), Jatrabari zone (JB) and at the end reaches and dumps the waste at Matuail dumping station (MT) within 1 AM.

5.5 Convergence Comparison

While implying both of the algorithms we found convergence in both cases. The values that we found and used to generate graphs for SARSA are given below:

SARSA	
Episodes	Reward
1	9.31
2	11.21
3	11.69
4	12.14
5	12.65
6	14.26
7	14.96
8	15.12
9	15.64
10	16.87
11	17.31
12	18.14
13	19.26
14	19.81
15	21.64
16	22.36
17	23.17
18	23.88

19	24.98
20	25.63
21	27.51

Table 4. Rewards for SARSA

The values that we found and used to generate graphs for Q-learning algorithm are given below:

Q-Learning	
Episodes	Reward
1	9.23
2	12.14
3	7.69
4	6.57
5	9.68
6	10.39
7	11.88
8	12.31
9	10.76
10	15.61
11	13.67
12	19.78
13	18.93
14	24.67
15	25.71
16	33.36
17	34.75
18	38.17

19	39.91
20	39.26
21	43.19

Table 5. Rewards for Q-Learning

The following figure shows us the early convergence of SARSA

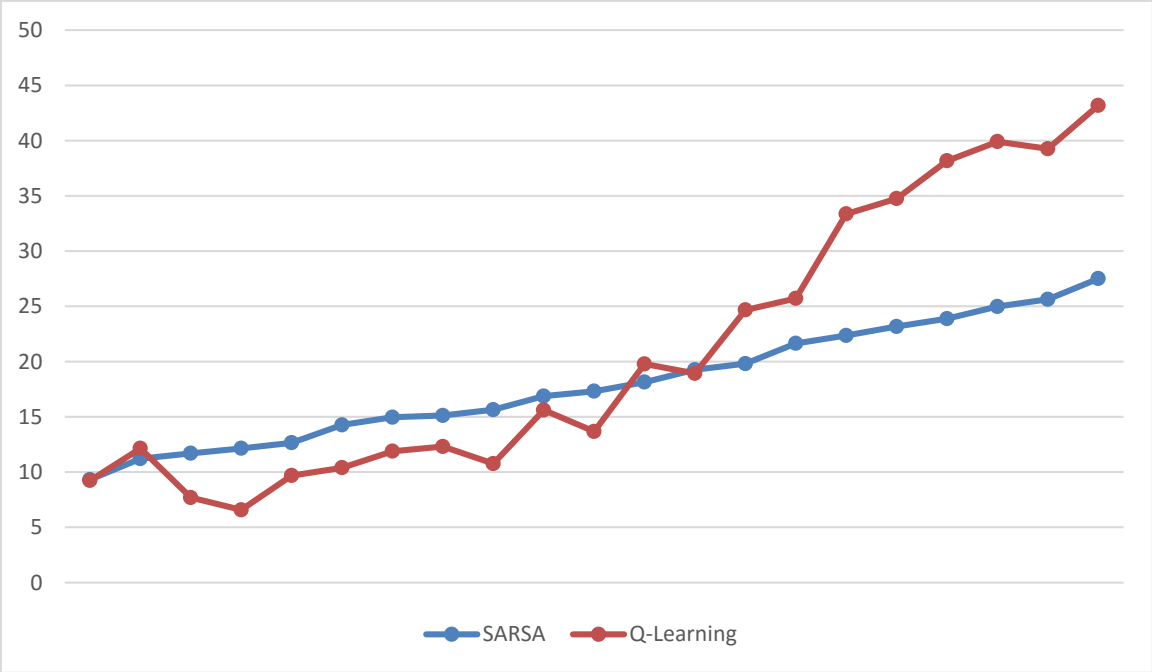

Figure 14: Early convergence of SARSA

Considering different results obtained from the reward of Q-learning and SARSA functions, we have come to the conclusion that SARSA has a faster convergence rate than Q-learning.

CHAPTER 6

6. Conclusion and Future Work

6.1 Conclusion

In this proposed model of ours we have taken into consideration the two algorithms, Q-learning and SARSA, in order to solve our problem, and observed results obtained from the model using a reward based machine learning algorithm. In the conclusion, we have showed the results and presented how fast our proposed model had worked in Dhaka city. We have seen that, in the end, the SARSA algorithm showed the best convergence speed, comparative to that of Q-learning. So, in case of our model, SARSA algorithm worked best.

6.2 Future works

In future, we wish to implement this model of ours in various other cities across the world where there will be tougher conditions in transportation and many other intricacies. We intend to develop our model and modify it to yield even better results for other scenarios, as well as implementing other algorithms of machine learning to provide better and more practically helpful solutions.

References

[1] Van Otterlo, M.; Wiering, M. (2012). "Reinforcement learning and markov decision processess". Reinforcement Learning. Springer Berlin Heidelberg: 3–42.

[2] Kaelbling, Leslie P.*;* Littman, Michael L.*;* Moore, Andrew W. *(1996).* "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research. 4: 237–285.

[3] Tokic, Michel; Palm, Günther (2011), "Value-Difference Based Exploration: Adaptive Control Between Epsilon-Greedy and Softmax", KI 2011: Advances in Artificial Intelligence , Lecture Notes in Computer Science, 7006, Springer, pp. 335–346, ISBN 978-3-642-24455-1

[4] Williams, Ronald J. *(1987).* "A class of gradient-estimating algorithms for reinforcement learning in neural networks". *Proceedings of the IEEE First International Conference on Neural Networks.*Watkins, Christopher J.C.H. *(1989).* Learning from Delayed Rewards *(PDF) (PhD thesis). King's College, Cambridge, UK.*

[5] Szita, Istvan; Szepesvari, Csaba (2010). "Model-based Reinforcement Learning with Nearly Tight Exploration Complexity Bounds" (PDF). ICML 2010. Omnipress. pp. 1031–1038.

[6] "Playing Atari with Deep Reinforcement Learning". Computing Research Repository. 1312.5602.

[7] Ng, A. Y., & Russell, S. J. (2000, June). Algorithms for inverse reinforcement learning. In Icml (pp. 663-670).

[8] Hastings, W. K. (1970-04-01). "Monte Carlo sampling methods using Markov chains and their applications". Biometrika. **57** (1): 97–109. ISSN 0006-3444.

[9] Kroese, D. P.; Brereton, T.; Taimre, T.; Botev, Z. I. (2014). "Why the Monte Carlo method is so important today". WIREs Compute Stat. **6**: 386–392.

[10] Gosavi, Abhijit *(2003).* Simulation-based Optimization: Parametric Optimization Techniques and Reinforcement. *Springer.* ISBN 1-4020-7454-9.

[11] Suita, Istvan; Szepesvari, Csaba (2010). "Model-based Reinforcement Learning with Nearly Tight Exploration Complexity Bounds" (PDF). ICML 2010. Omnipress. pp. 1031–1038.

[12] Coalfish, R. E. (1998). Monte Carlo and quasi-Monte Carlo methods. Acta Numerica. **7**. Cambridge University Press. pp. 1–49.

[13] A. Habib and M. I. Khan, "Reinforcement Learning based Autonomic Virtual Machine management in Clouds," in Proc. of 5th IEEE International Conference on Informatics, Electronics and Vision, Dhaka, Bangladesh, pp. 135-136, 201

[14] R.S.Sutton and A.G.Barto.Reinforcement Learning:An Introduction.TheMIT Press,Cambridge, Massachusetts, England, 2002.

[15] K.Gupta, "Performance Comparison of Sarsa(λ) and Watkin's Q(λ) Algorithms." (n.d.): n. pag. Print.

[16] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck, "From Data Center Resource Allocation to Control Theory and Back," in Proc. of 3rd IEEE Int. Conf. on Cloud Computing, pp. 410-417, Miami, FL, July 2010.

[17] T. Stockheim, M. Schwind, A. Korth, and B. Simsek, "Supply ChainYield Management Based on Reinforcement Learning"

[18] A. Habib, M. I. Khan and J. Uddin "Optimal Route Selection in Complex Multi-stage Supply Chain Networks using SARSA(λ)"

[19] Fakoor, Mahdi; Kosari, Amirreza; Jafarzadeh, Mohsen (2016). "Humanoid robot path planning with fuzzy Markov decision processes". Journal of Applied Research and Technology.