A HYBRID APPROACH FOR DYNAMIC TASK SCHEDULING IN UNFORESEEN ENVIRONMENTS USING MULTI AGENT REINFORCEMENT LEARNING AND ENHANCED Q-LEARNING

Jayakody Kankanamalage Chathurangi Shayalika

189394E

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

October 2020

A HYBRID APPROACH FOR DYNAMIC TASK SCHEDULING IN UNFORESEEN ENVIRONMENTS USING MULTI AGENT REINFORCEMENT LEARNING AND ENHANCED Q-LEARNING

Jayakody Kankanamalage Chathurangi Shayalika

189394E

Thesis submitted in partial fulfilment of the requirements for the

degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

October 2020

Declaration

I declare that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a Degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organization.

Name of Student — Signature of Student

J.K.C. Shayalika — Date: 30/10/2020

Supervised by

Name of Supervisor(s) — Signature of Supervisor(s)

Dr. Thushari Silva — Date: 30/10/2020

Dedication

I dedicate this thesis to my parents who are the pioneers and great pillars that reinforced my education and always supported and inspired me. This thesis is gratefully dedicated to all the lecturers of Department of Computational Mathematics, Faculty of Information Technology, University of Moratuwa.

Acknowledgement

I would like to express my sincere gratitude to my supervisor Dr. Thushari Silva who guided and profoundly nurtured me throughout the project. She extended her fullest corporation to me whenever I sought for advice. I'm also thankful to Prof. Asoka Karunananda who helped immensely from the beginning of the research. I was encouraged by his teaching methods to consider issues in multiple avenues.

I'm grateful to all the lecturers of Department of Computational Mathematics. During the course period of two years, these valuable lectures helped me to think differently.

I had to refer to several books and research papers as references for this research. I want to thank all the authors of these publications for their notable work. I'm especially thankful to *"Commonwealth of Learning-Canada",* online professional development program who assisted the fully access to online Specialization on Reinforcement Learning offered by *University of Alberta, Canada,* where I could grasp unparalleled knowledge on Reinforcement Learning.

Many thanks to all the editors and reviewers of the journals and conferences whom I submitted the papers related to the project, whose positive and negative feedback greatly assisted the progress of the research.

And, in many ways, my batch mates helped me a lot. I would like to give them my respect.

I want to express my deepest gratitude to my caring parents and sister for always helping me to pursue higher education.

Abstract

The process of assigning most appropriate resources to workstations or agents at the right time is termed as Scheduling. The word is applied separately to tasks and resources in task scheduling and resource allocation accordingly. Scheduling is a universal theme being conferred in technological areas like computing and strategic areas like operational management. The core idea behind scheduling is the distribution of shared resources across time for competitive tasks. Optimization, efficiency, productivity and performance are the major metrics evaluated in scheduling. Effective scheduling under uncertainty is tricky and unpredictable and it's an interesting area to study. Environmental uncertainty is a challenging extent that effect scheduling based decision making in work environments where environment dynamics subject to numerous fluctuations frequently.

Reinforcement Learning is an emerging field extensively research on environmental modelling under uncertainty. Optimization in dynamic scheduling can be effectively handled using Reinforcement learning. This research is about a research study that focused on Reinforcement Learning techniques that have been used for dynamic task scheduling. This thesis addresses the results of the study by means of the state-of-the-art on Reinforcement learning techniques used in dynamic task scheduling and a comparative review of those techniques. This thesis reports on our research on a Hybrid Approach for Dynamic Task Scheduling in Unforeseen Environments using the techniques; Multi Agent Reinforcement Learning and Enhanced Q-Learning.

The proposed solution follows online and offline reinforcement learning approaches which works on real time inputs of heuristics like, Number of agents involved, current state of the environment and backlog of tasks and sub-tasks, Rewarding criteria etc. The outputs are the set of scheduled tasks for the work environment. The solution comes with an approach for priority based dynamic task scheduling using Multi Agent Reinforcement Learning & Enhanced Q-Learning. Enhanced Q-Learning includes developed algorithm approaches; Q-Learning, Dyna Q+ Learning and Deep Dyna-Q+ Learning which is proposed as an effective methodology for scheduling problem.

The novelty of the solutions resides on implementation of model-based reinforcement learning and integration with the model-free reinforcement learning algorithmic approach by means of Dyna-Q+ Learning and Deep Dyna-Q+ Learning for dynamic task scheduling in an unforeseen environment. The research project also concentrates on how the dynamic task scheduling is managed within a constantly updating environment which the Deep Dyna-Q+ has provided a ground solution to cater this requirement. The end solution has comparatively evaluated the product using evaluation metrics in each of the three Q-Learning variations developed. As per the evaluation results it was revealed Deep Dyna-Q+ implementation would cater well the problem of dynamic task scheduling in an unforeseen environment.

Table of Contents

List of Figures

List of Tables

Abbreviations

DL	-	Deep Learning
RL	-	Reinforcement Learning
DRL	-	Deep Reinforcement Learning
QL	-	Q Learning
DQL	-	Deep Q Learning
DQN	-	Deep Q-Network
MDP	-	Markov Decision Process
MARL	-	Multi Agent Reinforcement Learning
DDQ+	-	Deep Dyna Q+ Learning
GPU	-	Graphics Processing Unit

Chapter 01
Introduction

1.1 Prolegomena

Task allocation means that the correct resources have been committed to implement and work item/task successfully for a particular result at the right time. It ensures that the demand for process implementation facilities is balanced against the availability of these services. Scheduling is a popular norm that is crucially being applied for numerous real-world applications like, industrial workforce management, web-server management, multiprogramming/ multitasking, grid computing, public transportation, traffic planning and network routing. Some other practical use cases of scheduling may include scenarios like waiting in a queue, waiting on hold for technical support, prioritizing a list of chaos, selecting jobs from a printer queue, balancing transmission of multiple signals over limited bandwidth etc.

The component called scheduler performs scheduling in a computer system, which mainly concerns throughput, latency and response time. Throughput refers to how easily a certain number of tasks per unit of time can be performed from beginning to end. In comparison, latency, which involves the waiting time before it can be served, is the processing time or the time it takes to complete the job from the time of request or submission until the end. Response time is the time taken to serve the procedure or order, in short, the waiting time.

Job-Shop Scheduling Problem (JSSP), Open-Shop Scheduling Problem and Flow-Shop Scheduling Problem [1],[2],[3],[4] are widespread conversed optimization problems in computer science, where ideal jobs are assigned to resources at particular times. A Job Shop is a work location in which several general-purpose workstations exist and are used to perform a variety of jobs. The simplest version of JSSP is that the n jobs J1, J2..., Jn of different processing times are provided, which need to be scheduled on m machines with different processing power while minimizing the potential makespan. The makespan is the cumulative duration of the schedule (that is, when production has been completed for all jobs) [3],[4]. Each task consists of a series of tasks that must be carried out in a specified order, and each task must be processed on a workstation or a machine.

With various scheduling algorithms that elegantly decide on the order of optimal resource allocation in multi-processing and multi-tasking systems, Computer Science based optimization has been well enriched. Among the most widely debated are First Come First Served (FCFS), Shortest-Job-First (SJF), Priority Scheduling, Round Robin (RR), Multilevel Queue Scheduling and Multilevel Input Queue Scheduling. Computer based scheduling solutions can be long term, short term or either medium term.

Scheduling solutions can be separated into two major groups as Static and Dynamic Scheduling [5],[6]. Decisions are made at compile time in Static Scheduling, while in Dynamic Scheduling (or adaptive job sharing), computational state data is used during execution to make decisions [5]. Static scheduling involves full advanced knowledge of task-set features that are difficult

to achieve in an unpredictable setting. Dynamic scheduling is more appealing than static scheduling since, at compile time, it can manage situations where dependencies are uncertain. It is more computationally challenging and facilitates multiple methods of parallelization, and requires dynamic load balancing [5]. Dynamic schedulers are also versatile and respond to a changing uncertain scenarios [6], which is suitable for implementation in an unpredictable environment settings. In this research, the researcher focuses on dynamic scheduling.

Enormous efforts have been made to resolve the question of complex scheduling of tasks. Among them, intelligent strategies such as machine learning, logic and resource planning from expertise, making rational decisions to continuously maximize resource planning in various process contexts are important [7]. At present, many Artificial Intelligence techniques have been used for dynamic task scheduling. The most popular techniques are Genetic algorithms, Artificial neural networks, Fuzzy logic are the most prominent [2],[8]. Reinforcement Learning is a new technology and a persuasive paradigm that has been tested to address the challenge of optimal complex activity preparation. It can be researched that the reasons behind reinforcement Learning to become a promising dynamic scheduling technology are its ability to deal with environmental instability/uncertainty in a dynamic environment, its ability to self-learn the environment, its computationally effective and highly adaptive. Reinforcement Learning with well-trained networks has been used in real-time scheduling [9]. By learning scheduling law, it can perform entirely different tasks. The ability to model multiple priority tasks at once is one of the major benefits of experience replay[10]. Reinforcement Learning encourages the feasibility of addressing the dilemma of complex task scheduling

1.2 Background and Motivation

Presently, as a result of falling computational resource prices, the availability of data and improved algorithms, Artificial Intelligence has reached the latest AI 2.0 growth period. In the AI 2.0 family, Deep Learning (DL), reinforcement learning (RL) and their combination-deep reinforcement learning (DRL) are representative strategies and comparatively advanced techniques [1]. DL is a subset of machine learning that initially resulted from a multi-layer Artificial Neural Network (ANN). The different structures of DL are Boltzmann Machine (BM), Deep Belief Networks(DBN), Feedforward Deep Networks(FDN), Convolutional Neural Networks(CNN), Recurrent Neural Networks(RNN), Long-Short Term Memory(LSTM) Networks and Generative Adversarial Networks(GAN) [1]. Among them are the most popular constructs of machine learning, CNN and RNN. CNN is widely used for dealing with spatial distribution data, whereas RNN is suited for managing time-series data [1]. A special variant of RNN that can study long-term dependencies is LSTM. GAN is a neural network branch that is used for generative modeling [11].

The field of machine learning that takes effective steps to optimize incentives/rewards in real situations is well known as Reinforcement Learning. Reinforcement Learning's aim is to optimize an agent's reward by considering operations in a complex environment. Under uncertainty, reinforcement learning can help sequential decision making [1],[7]. With only limited knowledge of the scenario and limited input on the accuracy of choices, the Simple

Reinforcement Learning algorithm works. Like the human brain, it is praised with good decisions and penalized for poor choices. Actor/Agent, environment, reward/incentive and actions are the four fundamental components of Reinforcement Learning. A player going through roles is an agent. An environment is a location the agent exists in. A reward is what the agent acquires after acting. The agent maintains a state in which the agent currently be. The ultimate goal of the agent is gaining as many rewards as possible. Popular RL algorithms are Q-learning, SARSA(State–Action–Reward–State–Action), DQN(Deep Q- Network), DDPG (Deep Deterministic Policy Gradients), NAF(Normalized Advantage Functions), A3C (Asynchronous Advantage Actor-Critic)[1],[11],[12]. In several real-world implementations, RL is implemented, including in robotics, UAV, multi-agent and smart traffic [12]. Best RL algorithms have gained high proficiency in domains, including board games (Go, Chess) and Atari games, with established and simple rules [11]. Using deep network architecture to forecast over 100 phases of potential frames effectively in the sense of Atari games [11]. It is acknowledged that reinforcement learning in robot handling has a bright future ahead of it. [11]. In Reinforcement Learning, study hotspots found include partial vision, Hierarchical RL, conjunction with other AI technologies and game theory. [12]. The integration of RL and other techniques such as genetic algorithms and neural networks is also a subject of this analysis.

The perception of Deep Learning and the decision making of Reinforcement Learning is been combined to a single segment through DRL [1]. DRL will also execute a number of tasks that include both precious interpretations of high-dimensional raw inputs and policy control. [1],[13]. AlphaGO, which beat Sedol Lee, the world Go champion, is a software developed by Google DeepMind following the DRL strategy. In research on DRL, there are several recent trends. Google Brain is one application that teaches robotic arms to unlock doors and pick things up on their own. Uber is now attempting to train Grand Theft Auto to manage real vehicles on real roads using DRL. A community of researchers has developed a Multi-Task Deep Reinforcement Learning Method for Scalable Parallel Task Scheduling (MDTS) in one of recent research [13]. But, when dealing with dynamic parallel computing environments and employment with various properties, DRL faces the curse of dimensionality dilemma for decision-making [13].

1.3 Problem in Brief

Optimized scheduling of tasks and resources has been a difficult and a challenging problem in uncertain and unforeseen settings. It has deficiencies in modeling the dynamics and uncertainty in the environments. Dynamic scheduling has been a crucial problem in industrial workforce management where the environment is uncertain. Ineffective scheduling leads to system/organization inefficiency, inaccurate decision making and analysis, resource wastage and high cost, thus degrade the overall organizational productivity. Therefore, there is a need of a precise solution which can handle above mentioned problematic situations.

1.4 Aim

Design and implement Dynamic Task Scheduling application for an unforeseen environment using Multi Agent Reinforcement Learning and Enhanced Q-Learning.

1.5 Objectives

In order to reach the aim, following objectives are identified.

i. Comprehensively review the literature review on Dynamic Task Scheduling, Multi Agent Reinforcement Learning, Deep Q-Learning and Dyna Q+-Learning.
ii. Design and develop hybrid approach by integrating Multi Agent Reinforcement Learning, Dyna Q+ Learning and Deep Dyna Q+-Learning for dynamic task scheduling.
iii. Implement the dynamic task scheduling using the proposed approach.
iv. Evaluation of the developed system.

1.6 Hypothesis

The hypothesis of the research is Multi agent reinforcement learning and Enhanced Q-Learning can be adopted to address the dynamic task scheduling problem in an unforeseen environment. Inspiration behind this hypothesis is coming from how the tasks are been allocated in a honeybee or an ant colony and how they maintain continuous workflow and balance in day to day schedules.

1.7 Solution

This research project aims to develop an optimized algorithm for dynamic task scheduling in an unforeseen environment. Multi agent reinforcement learning and Enhanced Q-Learning is used in order to achieve the outcome. The inputs for this process are environmental conditions and backlog of tasks. The outputs would be set of scheduled tasks and it results the optimal scheduled state of the environment.

1.8 Structure of Thesis

The research dissertation has been structured as follows.

An overall introduction to the research project has been given in **Chapter 1**. The prolegomena, context and motivation, research issues, purpose, aim, objectives, solution proposed are briefly explained here.

Chapter 2 aims to report the literature survey done on dynamic task scheduling with a comparative review of the current built algorithms and their merits and demerits**.**

Chapter 3 describes in depth the technologies which were implemented to establish the solution**.**

In **Chapter 4** the approach taken to solve the identified problems is presented through expounding the hypothesis, inputs, outputs, process followed, users, system requirements and the features of system.

With the system design, system architecture, modular architecture and simulation design premeditated, **Chapter 5** deals with the design of the proposed algorithm.

Chapter 6 is about how the established application was implemented by the researcher.

The evaluation performed on the established system is referred to in **Chapter 7.**

Chapter 8 will be continued with the future opportunities and enhancements identified and the thesis ends by presenting the conclusion and further work.

1.9 Summary

This chapter describes the complete picture of the whole project, showing the issue of research, aim, objectives, hypothesis and the novel solution. The next chapter will be on the literature review of the dynamic scheduling of tasks using Reinforcement learning and complications to describe the problem of study.

Chapter 02

Evolution and State-of-the-art of Reinforcement Learning in Dynamic Task Scheduling

2.1 Introduction

In chapter 01, we introduced the overall project. This chapter provides our critical analysis of studies using reinforcement learning methods on advances in dynamic task scheduling. We have structured this chapter under several headings, namely, Breakthroughs in Reinforcement scheduling, Research Findings, Challenges in Reinforcement scheduling and Problem definition.

2.2 Breakthroughs in Reinforcement scheduling

This section includes the previous research on dynamic task scheduling using reinforcement learning strategies. Firstly, for dynamic task scheduling, it discusses the most used reinforcement learning methods and then explains the standard reinforcement learning systems and frameworks developed so far.

Reinforcement Learning methods used for dynamic task scheduling are usually either one of the two primary categories; Model-Free or Model-Based. Model-Free Methods are two types of Value-Iteration Methods and Policy-Iteration Methods. As Value-Iteration Approaches, earlier researchers used Q-Learning, Deep Q-Network (DQN), Greedy Distributed Allocation Protocol (GDAP) and Monte Carlo Algorithm (MCA). As Policy-Iteration Techniques, Policy Gradient (DPG) Algorithms and Actor-Critic Methods have been commonly used. Another big group, Model-Based Approaches, can be split into two groups, Learn Model and Model Given. There is also the latest major category emerging today, the convergence between Model-Free and Model-Based Approaches. The Dyna-Q algorithm has its position in this group. The strategies described can be classified into the traditional grouping of improving learning as seen in Fig 1. These strategies have been defined as detailed below.

In the related literature, the dynamic task scheduling problem has been modeled as the Markov decision process (MDP). Markov decision-making is a mathematical model that explains the decision dilemma for an agent in the Markov property environment. Markov's property clearly says that, provided the present, potential acts are irrespective of the past [3],[10],[11],[14]. In the form of reinforcement, dynamic task/resource utilization decisions and maximizing long-term behavior based on delayed environmental incentives have been modeled as Markov decision processes.

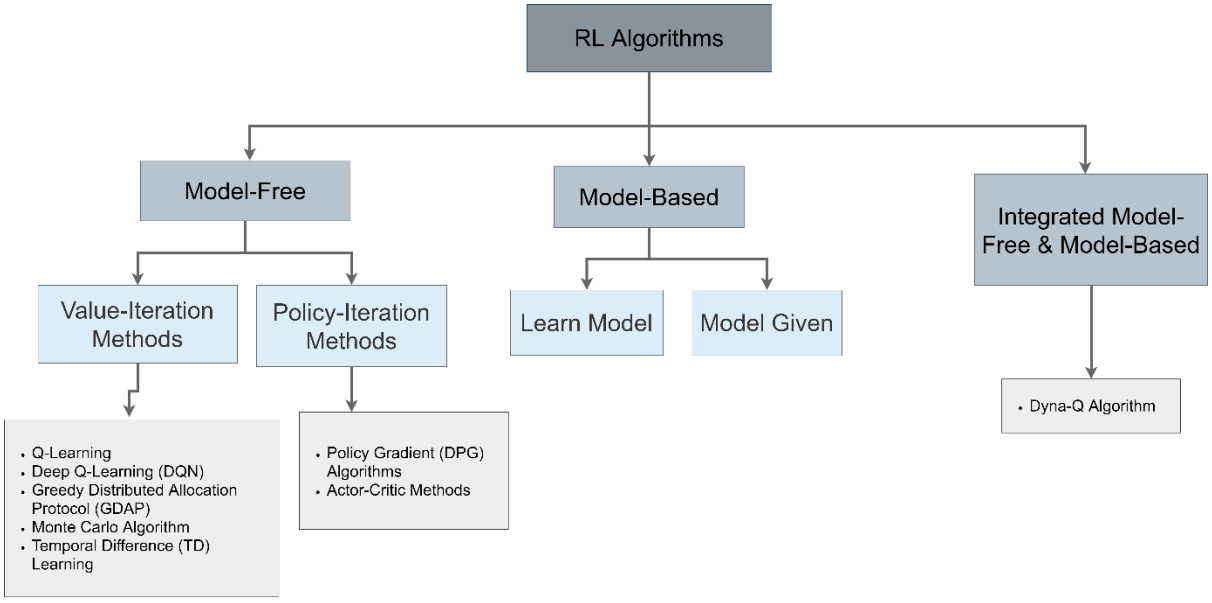


Figure 2.1 : Classification of Reinforcement Learning Techniques used in Dynamic Task Scheduling.

Source: Author

2.3 Model-Free RL

A Model-Free algorithm is an algorithm that calculates the optimal strategy without using or evaluating the dynamics (transition and reward functions) of the environment [11]. Model-Free algorithms function in the actual world in learning. In Model-free learning, the agent depends on trial-and-error practice to establish an optimal strategy. Value-iteration methods and policy-iteration methods are two primary techniques for representing agents with model-free reinforcement learning [11],[15].

2.3.1 Value-Iteration Methods.

Value-iteration methods upgrade the value function of the optimal policy on the basis of an formula (particularly Bellman equation) [11],[14]. These algorithms try to estimate the action-value function $\boldsymbol{Q(s, a|\theta)}$ for the optimal $Q*(s, a)$. The optimum policy is retrieved by

$$\boldsymbol{a = arg\ \max_a Q(s, a)}\ [11]$$

where 'a' represent the actions and s represent the states and 'Q(s,a)' is the Q value function of the state-action pair (s, a)

Value-iteration approaches are also carried out off-policy, meaning that the approach used to produce training data actions might be unrelated to the policy that is tested and enhanced, called the estimation policy [11],[12]. Popular value-iteration methods used in dynamic task scheduling are Q-Learning [7],[9],[10],[15],[16],[17] and Deep Q-Network (DQN) [3],[8],[18],[19],[20]. Apart from these two, Greedy methods [19], Monte Carlo Methods[21]

and Temporal Difference (TD) Learning [22],[23] also have been used. These methods have been listed in detail below.

2.3.1.1 Q-Learning

A Reinforcement Learning Based Dynamic Resource Allocation System (RLRAM) for Business Process Management(BPM) was proposed by Huang et al.[7] in their research. In this sense, Markov Decision Processes have been modelled on making decisions on the optimum distribution of resources and choosing which tasks need to be done depending on the incentive mechanism, Markov Decision Processes (MDPs). The proposed algorithm uses Q-learning to make allocation decisions in real time. In this article, the complex resource distribution optimization topic in BPM has been presented. Based on experiences with the environment, RLRAM can perform dynamic resource allocation. The optimum resource distribution problem in the implementation of a business process has been viewed as a sequential decision-making problem in this analysis. The proposed algorithm integrates with the context of the agent and discovers the best business process execution policies that will maximize the allocation of resources. The findings of the assessment show that the suggested solution responds well to established heuristics and techniques and can strengthen the existing state of business process management [7].

Xiao et al. [9] have proposed a distributed and self-adaptable scheduling algorithm focused on Q-learning for multiple work flows in real-time dynamic task allocation. This algorithm is not limited to adapting itself to the work arrival process (single task flow), but also takes full account of the influence of other agents on task flows (multiple task flows) [9]. To accomplish adaptive task allocation and node distribution, reinforcement learning is used here. This algorithm can increase the efficiency of the task and reduce the total per-task execution time. Arel et al. [16] used a Q-Learning algorithm using a feedforward neural network for value function approximation in their multi-agent and RL-based system for scheduling and achieving effective traffic signal management policy/traffic signals [16].

The research group Tian et el, in order to solve the hierarchical distributed task allocation problem of multi-robot systems in a complex dynamic environment, proposed a reinforcement learning approach based on the Q learning and Roulette process [15]. Multi-Robot Systems (MRS) have been used in their development to perform tasks that are difficult for an individual robot to perform, especially in the face of environmental and work-based uncertainties, remote control, missing knowledge and asynchronous computation. Two architecture aspects were taken up by the researchers; non-cooperation and cooperation, when designing the learning algorithm. In complex task allocation for self-adaptive fire disaster response, this suggested algorithm has functioned well. In a disaster response, the reward mechanism was initiated by considering the features of a Multi robot. In this way, three variables are correlated with the incentive function: the distance between the robot and the fire catastrophe, the intensity of the fire and the overall time. The simulation experiments were carried out on MulBotSim, which is a simulation framework for swarm robots built by Jilin University Control Theory and Intelligent Systems Laboratory [15]. In a fire emergency response, the suggested solution has accomplished successful multi-robot hierarchical mission allocation, because the fires have been eliminated in a timely manner.

A Fast Task Allocation (FTA) algorithm developed using Q-learning has been proposed by Zhao et al. [10]. Approximation of neural networks and prioritization of knowledge replay is used in the task allocation scheme. The mission allocation dilemma was modeled as an MDP by unpredictable transfer probabilities in order to provide a solution to the environmental ambiguity. In a study conducted by Wang and Usher [17] on manufacturing systems where reinforcement learning has been conducted on the topic of single machine dispatching rule collection, one more use of Q-learning is evident.

2.3.1.2 Deep Q-Network (DQN) / Deep Q-Learning (DQL)

To approximate the Q-value function, the Deep Q-Network (DQN) or Deep Q-Learning (DQL) leverages a Neural Network. In order to fix the curse of dimensionality in the regular Q-learning algorithm [3],[19], the Deep Q-Network idea was first introduced by Mnih et al. (2013). By explicitly taking the raw data (state features) as input and the Q function value of each state-action pair as output, DQN can handle complex decision making with broad and continuous state space [3].

Kumar et al. [8] implemented SchedQRM, which predicts a signature-based burst time of jobs and uses the Deep Q-Network algorithm to find an optimum solution for any arbitrary set of jobs. With custom states, behavior and incentives, the researchers created a full and complex work scheduling environment. With the help of two neural networks and a replay memory using the DQN algorithm, the machine agent investigates the context and learns an optimal strategy. Researchers use C++ object files for four programs to produce job signatures, namely: Matrix multiplication, Quicksort, Fibonacci sequence generator, and a random number generator, to create the data set for burst time estimation [8]. Finally, for all cluster load values, the group tested the proposed work against state-of-the-art Packer, Tetris, DeepRM and SJF (Shortest Job First), and concluded that SchedQRM performs superior or equal to the current heuristics.

The research done by Sun and Tan [18] focuses on the mobile crowdsourcing systems' trust-aware task allocation (TTA) optimization problem. This paper explores realistic and critical problems, including complex distribution of confidence-conscious assignments, which helps to increase the confidence score and minimize the cost of travel distance in uncertain and unpredictable crowdsourcing environments. Here, the topic of TTA optimization, which aims at optimizing the confidence score and reducing the expense of travel time, is formulated as Markov Decision Process Mobile Crowdsourcing (MCMDP). As an improvement over faith coordination optimization modeling in unpredictable crowdsourcing schemes, an enhanced Deep Q-Learning-based Trust Aware Task Allocation (Improved DQL-TTA) algorithm is proposed. The suggested DQL-TTA algorithm incorporates both optimization of confidence-aware task allocation and deep Q-Learning techniques. A theoretical analysis was done by the researchers to prove the applicability of the Improved DQL-TTA. The Deep Q-Network-based task allocation mechanism developed by the researchers is seen below.

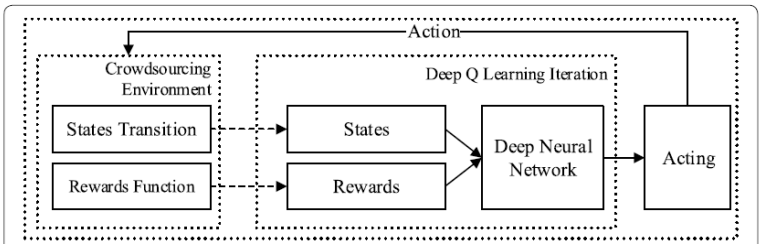


Figure 2.2 : Deep Q-Learning Driven Framework for Task Allocation. [18]

A dynamic task allocation mechanism for a dynamic context using cooperative Deep Q-Learning (TAP CDQL) has been suggested by Ben Noureddine et al. [19] to boost system efficiency using previous task allocation experiences. The new approach incorporates the strengths of unified/centralized and distributed approaches to learning. They have identified learning agents in their research that can teach each other how to assign tasks, making the allocation of tasks more relaxed and less difficult. In particular, if agents share their previous task allocation (exploitation) experiences to become a valuable understanding that can be used for future task allocation (exploration) processes, the overall system performance can be increased [19]. Agent contact or communication has been used intensively in their research. When an agent perceives a task, it follows a message protocol with its neighbors to perform the job it can probably perform. Deep Q-Networks (DQN) was used to allow agents to jointly coordinate their actions and automatically discover this protocol. Employing that, it was investigated by Ben Noureddine and others to enhance the efficiency and consistency of the decentralized allocation using insights from centralized approaches. As TAP CDQL, TAP and GDAP in JAVA have experimentally shown that the mechanism can handle the problem of task allocation in a dynamic real-time environment, the new method was implemented. The ability to handle heterogeneous agent types has not been completely exploited by researchers. In addition, their methodology also had some shortcomings due to the features of decentralization and reallocation, which were listed as future work in the study.

In Task Offloading for Mobile Edge Computing (MEC) applications, Deep Q-Networks also studied along with reinforcement learning. The role of unloading in a heterogeneous vehicle network consisting of several MEC servers, base stations, mobile vehicles and roadside units is investigated in [20]. In order to increase offloading reliability, the authors used Deep Q-learning technique for MEC server collection and transmission mode determination [20]. In a related study, DQN is suggested to resolve the issue of complex, flexible workshop scheduling primarily referred to as dynamic flexible job shop scheduling problem (DFJSP) [3]. Here, DQN advises the collection of suitable dispatching rules to reduce the overall wait in a complex, customizable workshop with new job inserts. The DQN is trained with two strategies reinforced by Deep Reinforcement Learning (DRL); double DQN and soft target weight upgrade [3].

2.3.1.3 Greedy Methods

Greedy approaches operate under the basic principle of preserving averages of action values, so that at every moment there is at least one action with the highest calculated value. In one of the study studies [19], Ben Noureddine et al suggested the Greedy Distributed Allocation Protocol (GDAP) which was used to resolve the complex issue of assigning of tasks in social networks

agents [19]. The key downside of the GDAP they found is that it depends exclusively on neighbor agents, which can trigger several unallocated tasks due to insufficient resources. The solution suggested in the same research [19] was: Task Allocation Method using Cooperative Deep Q-learning (TAP CDQL) [19], a new approach that seeks to solve this problem.

2.3.1.4 Monte Carlo Algorithm (MCA)

This approach involves a class of value-added learning methods that measure the value of a state by running a series of tests starting at that state and then combine the cumulative incentives obtained from the tests. Chantaravarapan et al. [21] have provided a complex scheduling method based on the Monte Carlo simulation. This approach confronts environmental uncertainties and is a cost-effective and easily scalable platform that can be used in the standard Excel spreadsheet environment [21].

2.3.1.5 Temporal Difference (TD) Learning

Temporal Difference (TD) learning is a reinforcement learning technique focused on a sequence of state transformations. In TD, this sequence is typically arranged in a set of trials, and each trial begins at some point and generally finishes in an absorbing state [23]. Temporal-Difference (TD) is a combination of the Monte Carlo (MC) method and the Dynamic Programming (DP) method. In a dissertation, Wei Zhang [23] explains how to implement reinforcement learning algorithms to discover strong domain-specific heuristics automatically for job-shop scheduling. Their research focuses on the topic of NASA space shuttle payload processing. The dilemma entails arranging a collection of tasks to fulfill a set of time and resource constraints, while at the same time minimizing the cumulative period of the schedule [22]. The TD (Temporal Difference) learning algorithm is used to train a neural network to learn a heuristic assessment function to select repair behavior over timeframes [22],[23]. This studied assessment function is used by a one-step search process to identify solutions to new scheduling problems.

2.3.2 Policy Optimization or Policy-Iteration Methods.

Policy-Iteration Methods or Policy-Based Methods estimate the utility function of a greedy policy arising from the most recent policy improvement. Policy-based approaches are also policy-based, which means that they estimate the importance of the regulation when using it for control [11],[14]. Policy-based methods parameterize the policy as;

$$\boldsymbol{\pi}\ \mathbf{(s,\ a|\theta),}$$

where 'a' represent the actions and 's' represent the states and 'π (s,a)' is the policy function of the state-action pair '(s, a)', and their target is to optimize θ either through gradient descent on an *objective function* **J(π)** or by maximizing local approximations of J [11] . The common policy-iteration methods used for dynamic task scheduling are Policy Gradient (DPG) Algorithms [24] and Actor-Critic Methods [24],[25]. These methods have been listed in detail below.

2.3.2.1 Policy Gradient (DPG) Algorithms

Policy gradient methods are another common choice for a variety of RL tasks. Policy gradient algorithms are classified into two groups of algorithms. There are Deterministic and Non-Deterministic Policy Gradient (DPG) algorithms. Lowe et al. [24] Deep deterministic policy gradients have been used in their studies on the application of a communication scheme of agents in Mixed Cooperative-Competitive Settings (DDPG). The analysis was generalized from a single agent to a Multi-Agent Deep Deterministic Policy Gradient Algorithm. Wei et al. [25] have developed a policy-based actor-critical algorithm approach to solve the problem of continuous-valued state and action variables in Consumer Scheduling and Resource Distribution in HetNets of Hybrid Energy Supply.

2.3.2.2 Actor-Critic Methods

In Actor-critical approaches, the actor is used to modify the parameters $\boldsymbol{\theta}$, where $\boldsymbol{\theta}$ indicates the model parameters for the policy function are used to determine the best response for a given state. A critic is used to test the actor's approximate policy function based on a temporal difference (TD) error. Lowe et al. [24] have suggested a multi-agent strategy gradient algorithm in which agents learn a centralized critic based on the findings and behavior of all agents. In one of the research studies, Wei et al. [25] used an actor-critical reinforcement approach to complex consumer planning and resource utilization in heterogeneous networks (HetNets). The solution was powered by hybrid technology and was planned to optimize the energy efficiency of the cellular networks. Liu et al. [26] is another research group that has been active in applying the Actor-Critic Deep Reinforcement Learning Application for the Solving Work Shop Scheduling Problem (JSSP). Their proposed model consists of an actor network and a vital network. Actor Network knows how to behave in diverse settings and work-based situations, while a vital network lets agents determine the meaning of the claims and then return the output to the actor network [26].

2.4 Model-Based RL

A Model-Based Algorithm is an algorithm that uses the transition / transformation function (and reward function) to approximate the optimum strategy. Model-based algorithms use a reduced number of real-world experiences during the learning process. The goal is to create a model based on the initial encounters, and then use this model to replicate more episodes, not in the actual world, but by adding them to a built model and returning the effects to that model. There are two types of Model-based RL in the literature; Learn Model and Model Given [11].

Marco Wiering [27] mentioned in his research multi-agent training algorithms to teach traffic light controllers to reduce the average waiting time of vehicles (cars) in urban cities. They also planned activities for each agent (vehicle) by treating the whole complex ecosystem as a multi-agent-based framework. The group has developed a series of multi-agent model RL traffic light control systems that can be used to refine driving policies for cars [27]. Compared to researchers working on complex task scheduling using Model-Free RL, fewer analysis attempts have been made using Model-Based RL.

2.5 Integration of Model-Free and Model-Based RL

Model-based reinforcement learning uses a reduced number of real-world experiences to learn. The issue with these methods, though, is that the model should be sharp and reliable to reflect the complex existence of the environment, which is difficult to do due to environmental variability. The convergence of Model-free and Model-based approaches is a recent research direction in Improving Learning that can solve complexity concerns. These strategies have the bonus of speeding up learning. Dyna-Q Learning algorithm is the most used integration technique in literature.

2.5.1 Dyna-Q Algorithm

Dyna Q Algorithm is an extension of the Q learning algorithm. Dyna –Q algorithm can integrate *model-free* **learning** and *model-based* **planning** capabilities. This algorithm was introduced by Richard S. Sutton in 1990 [14],[28]. Dyna architectures combine trial-and-error (reinforcement) learning and execution-time planning into a single mechanism that works alternately in the environment and on a learned model of the world [28]. Dyna-Q has been less studied earlier on complex job scheduling. Peng et al. [29]] have established Deep Dyna-Q as a research basis for Incorporating Mission Planning-Completion Dialog Policy Learning. In their design, an environment model, referred to as the world model, has been integrated into the dialog agent for the purpose of mimicking real user response and creating simulated experience. During the policy learning dialog process, the built world model is continually updated with actual user experience to approach real user behavior and, in turn, the dialog agent is configured using both real experience and virtual experience [29]. Su et al. [30] is another organization that has published studies focused on policy learning dialogues. They proposed a Discriminative Deep Dyna-Q (D3Q) approach to enhancing the reliability and robustness of Deep Dyna-Q (DDQ), a new paradigm inspired by the Generative Adversarial Network (GAN) that integrates a discriminator in the planning process [30].

2.6 Standard Dynamic Task/Resource Allocation Frameworks

Major literature on dynamic task scheduling has provided mechanisms for the distribution of services, which are specifically concerned with the control of the total dispersed tasks in the system scenario. The structures that have been built below.

2.6.1 MARL (Multi-Agent Reinforcement Learning)

Generally, task or resource distribution is done in a distributed environment where tasks/resources must be assigned to various machines/systems accordingly. Multi-Agent Reinforcement Learning (MARL) is the deep reinforcement learning discipline that focuses on the above-mentioned criterion. It involves models designed for multiple agents that learn by communicating dynamically with their surroundings. Whereas in single-agent reinforcement learning simulations the status of the environment varies exclusively as a function of the

behavior of the agent, in MARL, environmental issues are subject to the behaviors of all agents [16],[27],[31],[32],[33],[34].

Cui et al. [31] is a group of researchers who have developed a complex dynamic resource distribution system for a multi-UAV network through collaborative multi-agent and enhanced learning. The Multi-Agent Reinforcement Learning (MARL) based Autonomous Learner known as Independent Learner (ILs) algorithm was developed to address the overhead of knowledge sharing in multi-UAV networks. The goal of the Markov Decision Process (MDP) is to find an optimal maximum reward strategy. In order to understand the distribution of resources between UAVs, an agent-independent approach is proposed, for which all UAV agents independently perform decision algorithms but share a standard Q-Learning-based framework. Each UAV uses a typical Q-learning algorithm, which learns the optimum Q-value and agrees on an optimal MDP strategy at the same time.

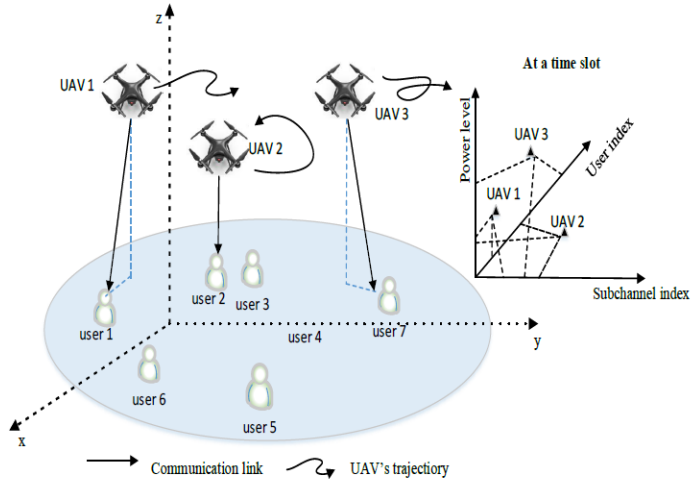


Figure 2.3 : Illustration of multi-UAV communication networks [31].

The simulation results have shown that the proposed MARL resource allocation algorithm for multi-UAV networks can achieve a trade-off between overhead information sharing and system efficiency. Another promising future path for study is mentioned by the researchers. This would include refining the operation and trajectory of UAVs in multi-UAV networks and would be able to further increase the energy efficiency of multi-UAV systems.

Arel et al. also implemented a multi-agent scheme and an RL-based mechanism for scheduling and obtaining an efficient traffic signal management strategy for intersection networks [16]. It also helps to minimize congestion and the possibility of cross-blocking intersections. SCOOT, SCATS, OPAC and RHODES are some of the available adaptive signal control systems [16]. This traditional deterministic traffic control schemes do not hold up to the scheduling of large signal networks in metropolitan environments, primarily due to the lack of a long-term incentive strategy. Five-intersection traffic network has been studied in this study, in which each intersection is controlled by an autonomous intelligent agent. Here, two types of agents have been hired, the central agent and the outbound agent. Outbound agents schedule traffic signals by following the Longest-Queue-First (LQF) formula, which has been shown to ensure reliability and fairness, as well as to work with the central agent by supplying local traffic

statistics. Researchers conducted simulations in the Matlab environment using a discreet case environment. Experimental results demonstrate the advantages of multi-agent RL-based control over remote LQF-controlled single-intersection control, thereby showing the way for effective distributed traffic signal control in complex settings.

Marco Wiering [27] also mentioned, in his study, multi-agent Reinforcement Learning (RL) algorithms in their traffic light control method where task scheduling has been done in the agent environment. Gabel and Reidmiller [34] have developed Adaptive Reactive Workshop Scheduling Approach with Enhancing Learning Agents for Output Scheduling Problems by modeling them as multi-agent Enhancing Learning Problems. By combining data-efficient batch-mode enhancement learning, neural network-based value function approximation, and using an optimistic inter-agent coordination approach, a multi-agent learning algorithm was developed in their research [34].

MAgent, a forum to promote research and the advancement of multi-agent preparation, has been launched by Zheng et al. [32]. MAgent focuses on supporting activities and systems involving, ideally, multi-agents, hundreds to millions of agents where previous testing platforms centered on single-agent reinforcement learning. It enables the study of learning algorithms for the optimal policies of agents in their development, in the interactions between the population of agents and, more significantly, in the observation and comprehension of the activities and social phenomena of individual agents emerging from AI society, including communication languages, leadership, altruism. MAgent is a highly scalable solution that can house up to one million agents on a single GPU server. The researchers developed three environments for live and immersive simulation, such as Chase, Gathering and Battle [32], which are versatile and can be personalized and used by AI researchers. Pursuit, Gathering, Battle shows the rise of local cooperation, collaboration in a scarce capital world and a hybrid of cooperation and competition. This study talks about Artificial Collective Intelligence (ACI) [32], where multiple AI agents work together, artificially generating their own collective intelligence. The goal of the proposed MAgent project is to develop a multi-agent reinforcement learning network for ACI science.

SchedNet is another multi-agent in-depth reinforcement learning system developed by Kim et al. [33]. There, agents learn how to plan assignments themselves, how to encrypt messages, and how to choose acts based on messages received. SchedNet can determine which agents may have the right to send encoded messages by knowing the value of the partly observed knowledge of each agent. SchedNet consists of three major sections; Actor, Scheduler and Critic.

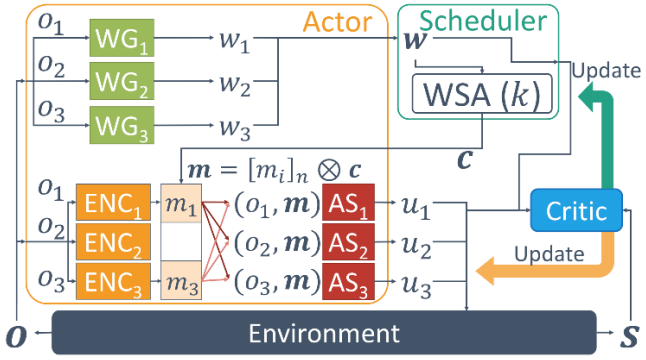


Figure 2.4 : The Architecture of SchedNet [33]

The SchedNet assessment based on several baselines for two separate programs, namely mutual contact and navigation and predator-prey [33]. Researchers also implemented three key components as scheduling, encoding and intervention where each block is a completely interconnected neural network. The distributed execution of SchedNet requires a WSA (Weight-Based Scheduler) where k agents are scheduled. These scheduled agents broadcast messages to all agents, and then behavior based on observation and received messages is chosen. The findings reveal a non-negligible efficiency difference between SchedNet and other processes such as vanilla scheduling approaches, such as round-robin, varying from 32% to 43% [33]. The solution is also capable of sophisticated scheduling, including smart scheduling and import-based scheduling.

2.6.2 OSL (Ordinal Sharing Learning)

Wu et al. [35] have suggested a novel multi-agent reinforcement learning system, the Ordinal Sharing Learning (OSL) method for work scheduling problems, specifically for the application of load balancing in grids.

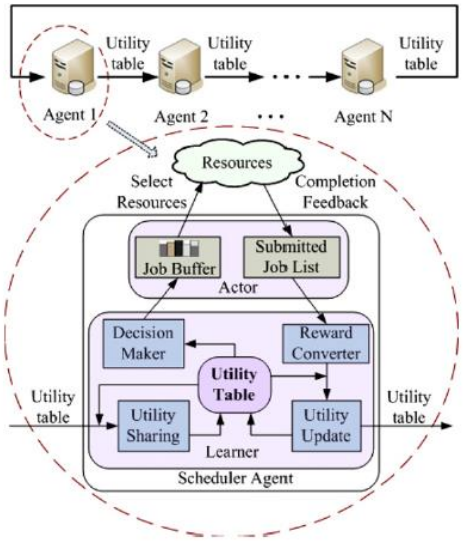


Figure 2.5 : The Schematic Diagram of the OSL Method for Job Scheduling [35]

The efficiency of the OSL algorithm applied is evaluated and compared to other algorithms, such as MARL. In order to solve the 'curse of dimensionality' problem in MARL, the researchers suggested this OSL algorithm, along with reduced computational complexity and improved coordination mechanism [35]. The researchers identified two facets of creativity as a simplification of decision-making in the scheduling of jobs and avoiding the issue of scalability and teamwork. By using a widely distributed learning technique, the approach eliminates the dilemma of scalability and requires multi-agent cooperation based on the information-sharing mechanism by limited communication [35]. The findings of the simulation show that the OSL solution can successfully achieve the load balancing goal and that its performance is also equal to other clustered scheduling algorithms [35]. The convergence property and the versatility of the proposed model are also illustrated. The future work of this analysis could include refining and using the suggested solution in real grid environments.

2.6.3 Gossip-Based Reinforcement Learning (GRL)

Gossip-based Reinforcement Learning (GRL) is another MARL-based scheduling system suggested by Wu and Xu [36] for clustered grid scheduling. A distributed scheduling framework based on multi-agent reinforcement learning is presented in the GRL method to enhance work scheduling and flexibility. The gossip system is designed to consider the autonomous organization of decentralized schedulers. The simulation results demonstrate that the proposed GRL-based schedulers can effectively complete the grid job scheduling mission and effectively accomplish load balancing with average contact costs.

GRL has two developments relative to other MARL-based scheduling schemes. First, each scheduler adopts a clustered, bandit-like GRL reinforcement learning algorithm. [35],[36]. In order to determine the current performance and load of services, a utility table is learned only on a scale with a resource number and a decentralized scheduling strategy can be derived directly from the utility table. So, the GRL is effective in computing. Second, through a gossip mechanism, proper coordination is accomplished [35],[36]. The gossip function helps each scheduler to communicate randomly with its neighbors. As compared to Ordinal Sharing Learning (OSL) [35] and Condor Flock P2P [36] Schedulers, GRL-based Schedulers can then operate without relying on special Schedulers. Likewise, the GRL is relaxed and robust in unison.

2.6.4 Centralized Learning Distributed Scheduling (CLDS)

Moradi [37] has introduced a multi-agent system for job scheduling in Grids, called Centralized Learning Distributed Scheduling (CLDS), with the use of reinforcement learning systems. This is a model-free approach that uses work information and completion time to approximate resource efficiency. There is a learner agent and several scheduler agents in this method that perform the task of learning and scheduling work using a coordination strategy that keeps the cost of communication at a low level. The researchers measured the reliability of the CLDS method by modeling and performing a series of tests on a virtual grid structure under various system sizes and loads. The findings demonstrate that even in large and massive loaded grids,

CLDS can efficiently balance the load of the system while preserving its adaptive performance and scalability.

2.7 Research Findings

This section provides a comparative study of the Reinforcement Learning approaches used for complex dynamic task scheduling in early studies. In the following table, the advantages and demerits of the methods mentioned have been discussed in terms of the complex task plan.

Table 2.1 : Comparison of Reinforcement Learning Techniques used for Dynamic Task Scheduling.

Technique	Merits	Demerits
Q-learning[7],[9], [10],[15],[16],[17]	-A powerful algorithm. -Have attained significant positive results. - Simple to configure & fine-tune the parameters.	- Q-learning is confronted by the intrinsic non-stationary nature of the environment. [24]. -Lack of generalizability. -Q-learning agents solely does not have the ability to approximate values and decision making for unseen states. - Defining a Q-table will be a very complicated, time-consuming and demanding process, particularly when state space is very large.
Deep Q-Learning [3],[8],[18]–[20]	- A great success has been achieved in the high dimensional problem.	-Discrete action space.
Greedy Methods [19]	-Adaptive in many situations.	-Especially in reference on neighbor agents, which can create several tasks unallocated due to insufficient resources.
Monte Carlo Method [21]	-Have the ability of reducing the makespan of jobs by applying probability-based distribution approach [21].	-Possible limits in Dynamic scheduling. - Generally, may leads to noisy gradients and high variance [21].
Temporal Difference (TD Learning)[22],[23]	- Faster learning [22]	- TD learning is usually less stable and can lead to inaccurate solutions.
Policy Gradient Algorithms [24]	- Have stronger properties for convergence. - Particularly efficient in high-dimensional space. - A stochastic strategy can be learned, while value functions (Q learning) cannot.	-Policy gradient struggles from a variance that rises as the number of agents increases [24]. -Slow convergence, since this takes a lot of time to train.
Actor-critic [24], [25]	-Consider the intervention/action policies of other agents so that they can effectively learn policies that involve complex multi-agent teamwork.	

Model-based RL Algorithms [11], [27]	-Reusability of the models. -Speed up the agent learning process.	- Have more hypotheses and approximations regarding a task but could be confined only to certain unique types of tasks. -The model must be realistic enough to pose a challenge in the real world. If the model is wrong, there is a high chance of discovering something different from reality.
Dyna-Q Algorithm [14],[28],[29], [30]	-Speeding up the learning -Can showcase positive remarks to solve environment uncertainty.	-Limitations in solving environment uncertainty when action-space and observation space becomes larger.

The implemented dynamic task scheduling frameworks can be evaluated as below table.

Table 2.2: Comparison of Reinforcement Learning Based Task Scheduling Frameworks

Framework	Merits	Demerits
MARL[16],[27], [31],[32], [33],[34].	- Able to resolve the overhead of the sharing of information and computing [31]. - Able to achieve a trade-off between overhead information sharing and device efficiency [31]. - Allow the sharing of tasks where agents may seek assistance from cooperating neighbors.	-'Curse of dimensionality' problem [35] -Training or organizing training through a variety of agents is challenging. -Ambiguity-The MARL models are very sensitive to agent ambiguity scenarios. -May be reluctant to interact with heterogeneous agents. -Computational shortcomings surrounding the features of decentralization and reallocation. -Experiments mostly on small spaces in state-action.
OSL [35]	-Reduced computing complexity. -Improving the teamwork process and coordination among agents involving. -Better decision-making in the planning of jobs. -Avoidance of scalability and coordination issues. – Strong Convergence. -Adaptiveness. -Good approach to avoid the dilemma of scalability [35].	-Dependence on specialized schedulers. -Has not yet been tested in the real-world environment settings.
GRL [35],[36]	-Can successfully complete the mission of grid scheduling and achieve effective load balancing.	- Still not tested for real grid problems.

	-The gossip system allows each scheduler to interact arbitrarily with its neighbors. So, strong coordination has been accomplished. -Effective in computing. -Agents are robust in collaboration with other agents. -May obtain timely model knowledge and enhance efficiency through autonomous coordination. - Adaptive, robust scheduling. - Could cope with the variety of applications and diverse grid environments [36].	- Further Works-Testing the robustness of the algorithm in evolving network conditions [36].
CLDS [37]	- Performs well in load balancing and converges, in some situations, to a sub-optimal policy or even to a near-optimal policy [37].	-Have restrictions in scalability and computational performance.

2.8 Challenges in Reinforcement scheduling

Over the years, a broad variety of research has been performed on complex work management. Reinforcement Learning has, above all, been able to provide effective testing guidance through perfect task scheduling. The optimized and complex scheduling of tasks in unpredictable conditions is yet to be a difficult problem. The technologies introduced retain low efficiency in the execution of activities owing to environmental uncertainties. Dynamic task scheduling in an unpredictable and ever-changing environment, where many agents are interactively involved, has been less investigated by early work. There are many generalization problems encountered by a variety of study organizations. The inappropriate time of computing for real-time execution and the difficulty of algorithms is also a matter of concern. Owing to the complexity, most algorithms have not been able to solve real-world problems. Adaptive task scheduling in complex and unpredictable settings remains a research problem.

2.9 Problem Definition

Having considered the summary of issues identified in dynamic task scheduling using Reinforcement Learning, we have noticed that little research has been conducted to improve optimization of task scheduling in an unforeseen environment using model-based approach. We also notice many researchers have used only single approach across the domains and they have many generalization issues. Based on our critical review, we define our research problem as how to schedule dynamic tasks efficiently in an unforeseen complicated environment.

It was able to gather ample evidence on the applicability and limitations of the domain since the state-of-the-art advances in task scheduling using Reinforcement Learning. In the same

way, some inspiring clues that can be employed to overcome the identified drawbacks of scheduling using Reinforcement learning also could be gathered.

2.10 Summary

In this chapter we have given a critical review of reinforcement learning by highlighting applications of dynamic task scheduling and the limitations / issues in the reinforcement learning techniques. We have also identified various technologies used for dynamic task scheduling. More importantly we have defined our research problem as how to schedule tasks efficiently in an unforeseen environment. In the literature review we have also identified multi agent reinforcement learning and Deep Q learning technologies as potential technologies for dynamic task scheduling. In chapter 03 we discuss our methodology for use of multi agent reinforcement learning and Deep Q learning for dynamic task scheduling.

Chapter 03

Technology

3.1 Introduction

In chapter 02, a critical review of literature review of the use of reinforcement learning in dynamic task scheduling has been presented. The literature review identified the research problem and potential technology for solving the research problem. As such, this chapter presents the technology adopted in our solution to dynamic task scheduling in an unforeseen environment using multi agent reinforcement learning and enhanced Q-learning.

3.2 Reinforcement Learning

One of the three major sub-areas of Machine Learning is reinforcement learning. Reinforcement Learning is the science of using experience to make optimal decisions. As humans, we are most familiar with the nature of learning. We interact with our environment, perform some actions, get feedback from the environment. The feedback is accompanying with a reward or a penalty, meaning that we are appreciating for each positive action we are doing and penalized for every negative action, both of them giving a glimpse of what has to be done and not to be done in log run. Somehow, we learn from mistakes and makes thoughts by determining not to perform faults again. It does not have an explicit instructor when a child plays, waves his arms, or looks around, but it has a direct sensorimotor connection to its environment. In order to maximize a numerical reward signal, reinforcement learning problems involve learning what to do and how to map situations to actions. [14].

Reinforcement Learning process includes the following simple steps:

- Environment observation
- Using some tactic, determining how to act.
- Behaving accordingly
- Achieving a reward or penalty
- Learning from experiences and refinement of our tactic/approach
- Iterate until it considers an optimal plan

3.3 Elements of Reinforcement Learning

3.3.1 Agents

The agents include the learner and the decision maker. Based on the rewards and punishments, agent takes the future decisions.

3.3.2 Actions

Actions includes series of activities which can be performed by the agent.

3.3.3 States

The condition of the agent in the environment.

3.3.4 Environment

The environment is the venue where the agent discovers and determines which actions to perform.

As discussed in Sutton and Barto, Reinforcement Learning: An Introduction [14] , beyond the above 4 components, other four main sub elements of a reinforcement learning system can be identified: a policy, a reward signal , a value function, and optionally, a model of the environment.

3.3.5 Policy

A policy determines the way of behaving of the learning agent at a given time [14]. Simplest terms, a strategy is a mapping of the behavior to be taken in certain states from perceived environmental states. In the sense that it alone is necessary to evaluate behavior, the policy is the cornerstone of a reinforcement learning agent. Policies can be stochastic in general.

3.3.6 Reward Signal

In a reinforcement learning problem, a reward signal determines the target. In each step, the environment sends a single number, a reward, to the reinforcement learning agent. The sole aim of the agent is to optimize over the long run the cumulative reward it earns. The reward signal thus decides what the good/positive and bad/negative occurrences of the agent in that specific environment are.

3.3.7 Value Function

In the long run, the value function determines what is fine. Mainly, the value of a state is, starting from that state, the cumulative amount of reward an agent can expect to accumulate over the future.

3.3.8 Environment Model

The model of the environment imitates the real environment's actions. It allows inferences to be made about how the setting would behave. For planning, models are used [14]; by which we mean some way to decide on a course of action by imagining potential future scenarios before they are actually encountered.

3.4 Markov Decision Processes

The problem of complex task scheduling can be generally modeled as a Markov Decision Process (MDP). A mathematical model to explain the decision problem for an agent in an environment with the Markov property is the Markov decision mechanism. The Markov property simply states that, given the present [3],[10],[11],[14], the future acts are independent of the past. A mathematical basis for defining an environment in reinforcement learning is the Markov Decision Process (MDP).

3.5 Model-Free RL & Model-Based RL

The methods of reinforcement learning used for dynamic task scheduling mainly belong to any either of the two main categories: model-free or model-based. Value-Iteration Methods and Policy-Iteration Methods are two types of model-free methods. The former researchers used Q-Learning, Deep Q-Network (DQN), Greedy Distributed Allocation Protocol (GDAP) and Monte Carlo Algorithm (MCA) as Value-Iteration Methods. Policy Gradient (DPG) Algorithms and Actor-Critic approaches have been commonly used for Policy-Iteration Methods. Model-Based approaches can be split into two groups, such as Learn Model and Model Given, another major category. Integration of Model-Free and Model-Based Approaches is also the third big category that is emerging today. The algorithm for Dyna-Q has its position in this group. Following section leads layout to describe technical aspect of each of these techniques.

3.6 Model-Free RL

A Model-Free algorithm is an algorithm which estimates the optimal policy without the dynamics (transition and reward functions) of the environment being used or evaluated [11]. In order to understand, model-Free algorithms function in the real world. In model-free learning, to set up the optimal strategy, the agent relies on trial-and-error experience. Value-iteration methods and policy-iteration methods are two key approaches for representing agents with model-free reinforcement learning [11],[15]. RL approaches based on value iteration have been used in the solution; these are Q-learning and Deep Q-learning.

3.6.1 Q-Learning

Being a model-free reinforcement learning algorithm, Q-learning is a commonly used algorithm that is based on values. Value based algorithms, based on an equation, update the value function (particularly Bellman equation). Q-learning is an off-policy learner. It implies that the value of the optimal strategy is discovered independently of the behavior of the agent. The 'Q' stands for quality or the consistency in Q-learning. Quality is the utility of a given action in achieving a potential reward.

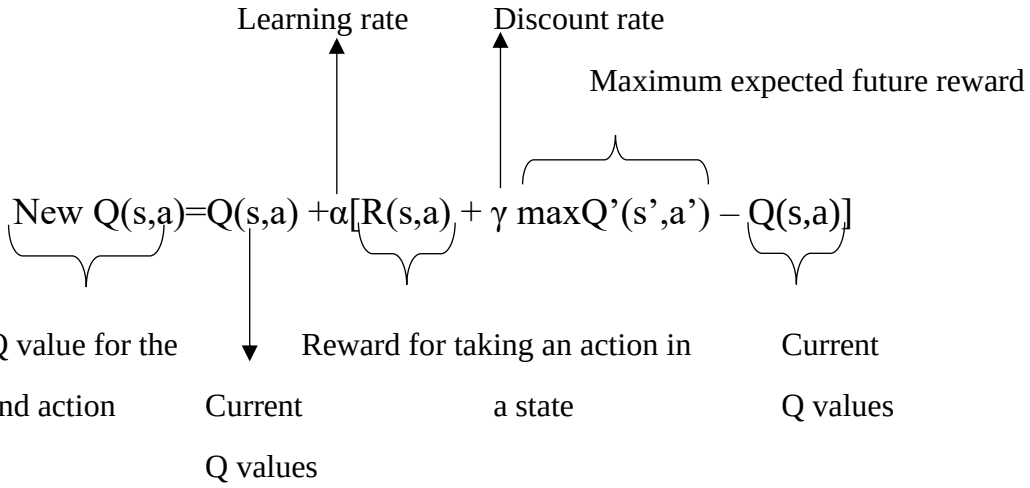
Q-learning Algorithm Definition:

i. Q*(s,a) is the estimated/expected value of doing an action 'a' in state 's' (cumulative discounted reward) and then implementing the optimal strategy.
ii. To estimate the value of Q*(s,a), Q-learning uses Temporal Differences (TD). An agent learning from an environment through episodes without previous experience of the environment is known as Temporary Difference.
iii. A table of Q [S, A] is maintained by the agent, where S is the set of states and A is the set of actions.
iv. Q [s, a] reflects the estimate of the latest/current Q*(s,a) calculation.

$$Q^{\pi}(s_t, a^t) = E\,[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid s_t, a_t]$$

Q-values for the state given a particular state — Expected discounted cumulative reward — Given the state and action

Bellman Equation



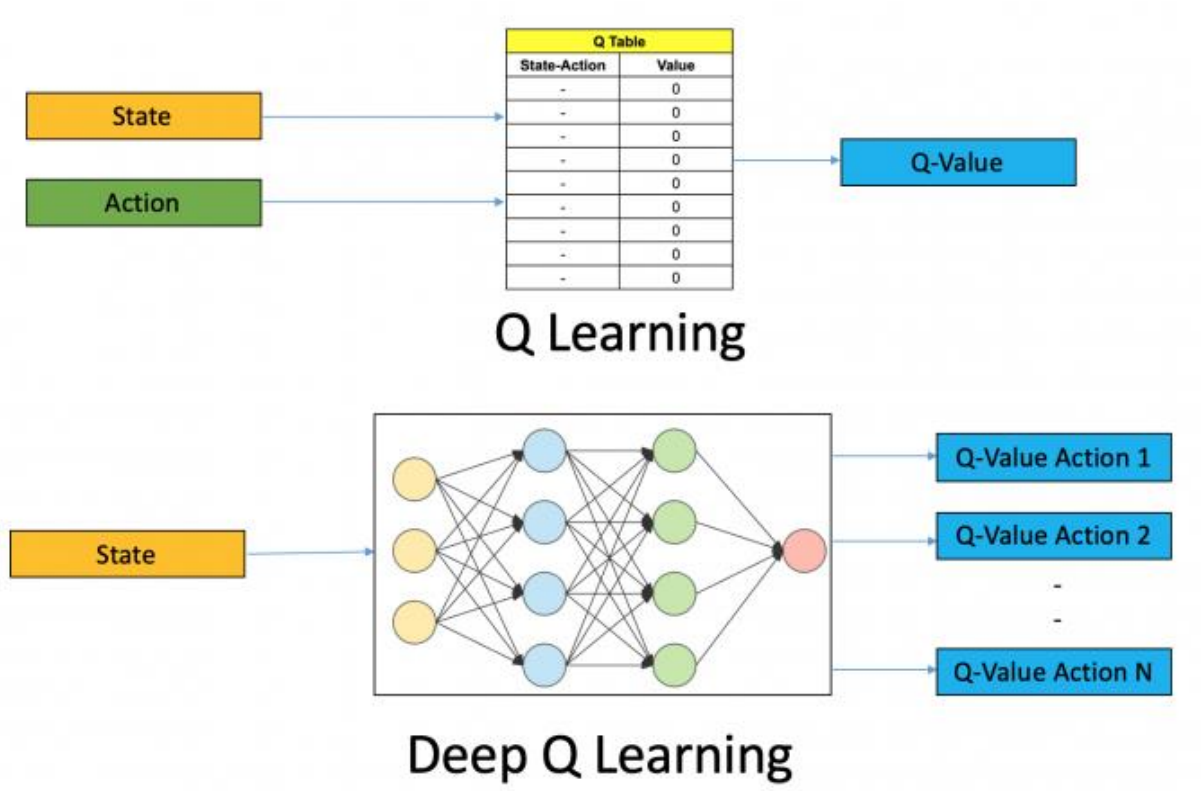
Bellman Equation Explanation for the episodes

When there is a relatively simple and comparatively straightforward environment to solve, Q-learning performs better, but when the number of states and actions we can take becomes more complicated, it is important to incorporate deep learning as a function approximation.

3.6.2 Deep Q-Learning

Deep Q-Learning is the art of incorporating artificial neural networks into the Q-learning algorithm, and a network that uses the neural network to approximate Q-functions is known as a Deep Q-Network (or DQN). The algorithm for DQN was developed in 2015 by DeepMind. The algorithm was first created by enhancing standard Q-Learning with deep neural networks and replay of experience (experience replay).

Firstly, in a DQN, the interactions of a single agent in an environment are preserved by the agent in memory. Once a certain limit or simply a threshold is reached by the agent, then we tell the agent to learn and improve from it. So, from a batch of interactions, the agent can now learn and understand. The agent randomly selects a uniformly distributed sample from this batch from these interactions and learns from that. Each experience is defined by the state in which it was, the action taken, the state in which it ended, and the reward it collected. This technique suitable for applications that have larger state space [3],[19]. The state is provided as the input, and as the output, the Q-value of all possible actions is generated. The main algorithmic difference between Q learning vs Deep Q learning can be shown as in Figure 3.1.



1

Figure 3.1: Main Algorithmic Difference between Q learning vs Deep Q learning

[1] https://www.mlq.ai/content/images/2019/07/deep-q-learning.png

In DQN, the target values are calculated from the right hand part of the Bellman Equation $Q^*(s,a) = E [R t + 1 + \gamma \max a' q*(s',a')$. Then, the outputs of the model are compared to the target values and loss is determined. Then, the neural network is updated, and its values are obtained using a backpropagation algorithm and stochastic gradient descent to minimize the error. In most DQN implementations have used two Q networks, which is not compulsory, but having two networks optimize the performance and solves the problem of moving target. Moving target is a problem that occur when agent estimates Q values and target value using the same neural network. Hence, to avoid moving target problem, generally another Q network (target network) with fixed parameters is used and the target network is updated periodically.

3.7 Model-Based RL

A model of the environment's behavior is established in Model-Based RL to reflect the real environment. A smaller number of connections with the real world are used by model based RL applications. The aim is to create a model based on initial interactions in the environment and then to simulate more episodes using this model. In Model-based RL, the model may be known or learned (Learn model vs Model Given). In order to resume learning, there is no need to wait for the environment to react or reset the environment to some state, and thus, model based RL has the benefit of speeding up learning.

3.8 Integration of Model-free RL & Model-based RL

3.8.1 Dyna –Q Algorithm

Dyna Q is a powerful algorithm merging Q-learning and Q-planning where, while collaborating with the environment, planning is done online. Models are used here to produce a policy and enhance it. To approximate the standard optimal control method known as Dynamic Programming (DP), Dyna architectures use machine learning algorithms [14],[28]. Trial-and-error reinforcement learning and execution-time planning are combined by Dyna architectures into a single mechanism that operates alternately on the environment and on a learned world model. Real or simulated experience improves both the model via model learning and value function/policy via direct RL. Model contains state, action, next state and reward tuples. Thus, the model can be both improved and queried to get to the next state in planning part [38]. In Figure 3.2, the possible ties between knowledge/experience, model, values, and policy are summarized. The overall architecture of Dyna Q algorithm which is described in the reference book, “Reinforcement Learning: An Introduction” by Richard S. Sutton and Andrew G. Barto [14], is shown in Figure 3.3.

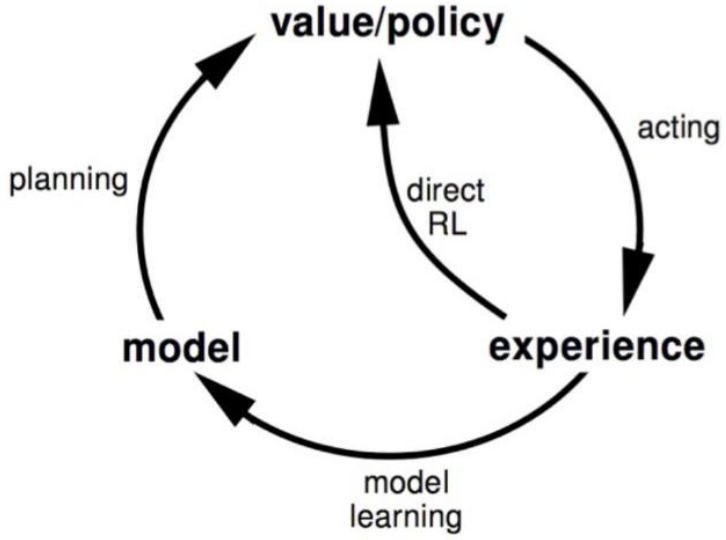


Figure 3.2: Relationship Among Learning, Planning and Acting [14]

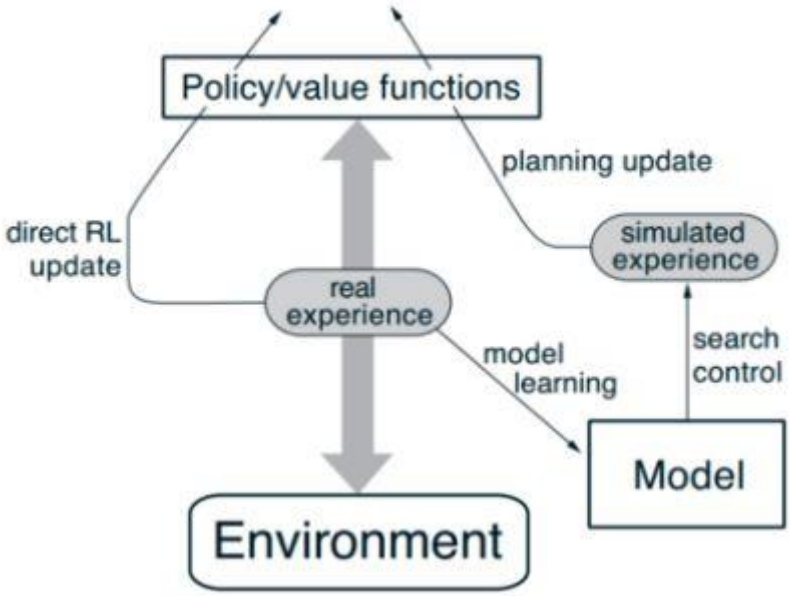


Figure 3.3: The General Dyna Architecture [14]

3.8.2 Dyna – Q+ Algorithm

The Dyna Q+ algorithm is an upgrade to the Dyna Q algorithm, allowing the agent to be able to explore new states in order to adapt to the evolving environment. The algorithm is completely the same as Dyna-Q, apart from the action of Dyna-Q, it keeps track of the amount of time a state was visited and gives reward to a state that has not been visited for a long time. Dyna-Q+ is more likely to detect the changing environment by offering additional rewards to the un-explored states.

3.9 Action Selection Mechanisms

Action Selection Mechanisms are being utilized in Reinforcement learning modelling in order to cater with [1]exploration vs. exploitation dilemma. These algorithmic processes allow decision-making agents to make a tradeoff between taking advantage of what they already know about the world or exploring further. As explained well in the reference book "Grokking Deep Reinforcement Learning" by Miguel Morales [39] and the [1]Reinforcement Learning Coursera Specialization the strategies can be summarized as below.

Greedy- In this sort of choice of action, the action with the highest estimated value is always chosen as the next action. Generally, it's not considered as a smart strategy, since we would quickly get stuck in a local maximum.

e-epsilon- In the beginning of reinforcement learning, an exploration rate is defined as "epsilon," which is normally set to 1.

Epsilon-Greedy- Select the action with the highest importance/value almost always. This is considered as one of the simplest ways of balancing exploration and exploitation. It is done greedily most of the time and explore, every so often, by acting randomly. Here, a uniform random number is taken (delta), and if delta is less than a small constant value, (epsilon), then an action is picked randomly from the set of all available actions for a given state.

e-epsilon decay- The basic idea is, first maximize exploration, then maximize exploitation. Here the epsilon is being reduced while the agent learns. Here, it produces a random number. Exploration and exploitation are performed based on the criteria specified. If this random number is greater than the epsilon, exploitation is performed (this means we use what we already know to select the best action at each step). If the random number is less than epsilon, exploration is done. The idea is that we must have a significant epsilon at the beginning of the Q-function learning. Then, gradually reduce it as the agent becomes more confident estimating Q-values. The procedure is well explained graphically in figure 3.4.

Softmax – Here, as a graded function of expected value, the action probabilities are decided. The most popular softmax approach uses a distribution of Gibbs, or Boltzmann. The selection of Softmax action stops the exploration automatically after some time.

[1] https://www.coursera.org/specializations/reinforcement-learning

[1]

Figure 3.4: Exploration and Exploitation behavior in e-epsilon-decay

3.10 Multi Agent Systems

An agent is something that perceives the environment through sensors and acts on the environment through actuators, according to Russell & Norvig. The multi-agent system (MAS) is a series of many autonomous/intelligent agents, each acting towards its goals while all interacting in a common world, being able to communicate and possibly organize their actions.

Multi agent system is a model of how teamwork can produce smart solutions for real world problems. Technically, MAS is loosely coupled network where the connections are not permanent. MAS consider the intelligence as an emergent feature. When designing agents in developing a MAS, the environment can be either; fully observable or partially observable, deterministic vs stochastic, episodic vs sequential, dynamic vs static, discrete vs continuous and single agent vs multi agent. There are three essential features that every multi agent system should have; known as communication, coordination and negotiation.

[1] *https://cdn-media-1.freecodecamp.org/images/1*9StLEbor62FUDSoRwxyJrg.png*

3.11 Single Agent Reinforcement Learning and Multi Agent Reinforcement Learning

The state of the world changes only because of an agent's behavior in single-agent reinforcement learning scenarios. It can be shown in figure 3.5.

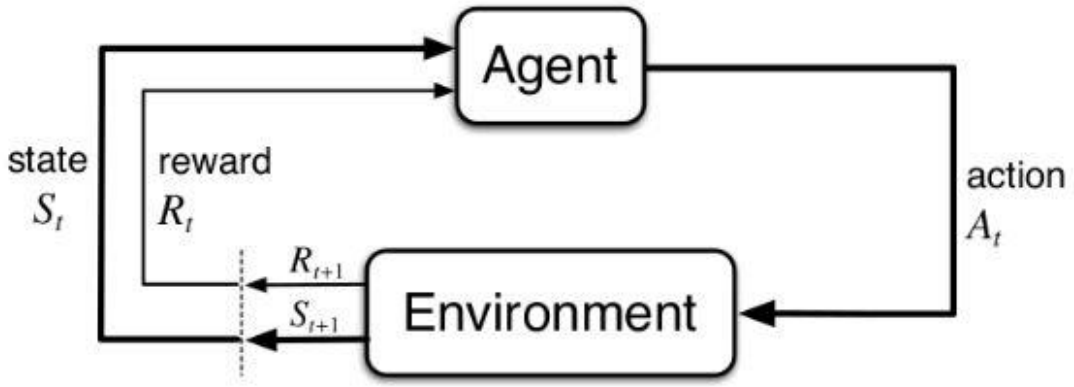


Figure 3.5: Single Agent RL [14]

The world is exposed to the behavior of all agents in MARL scenarios. So, with the number of agents in the environment, the complexity of MARL scenarios increases. It can be shown in figure 3.6.

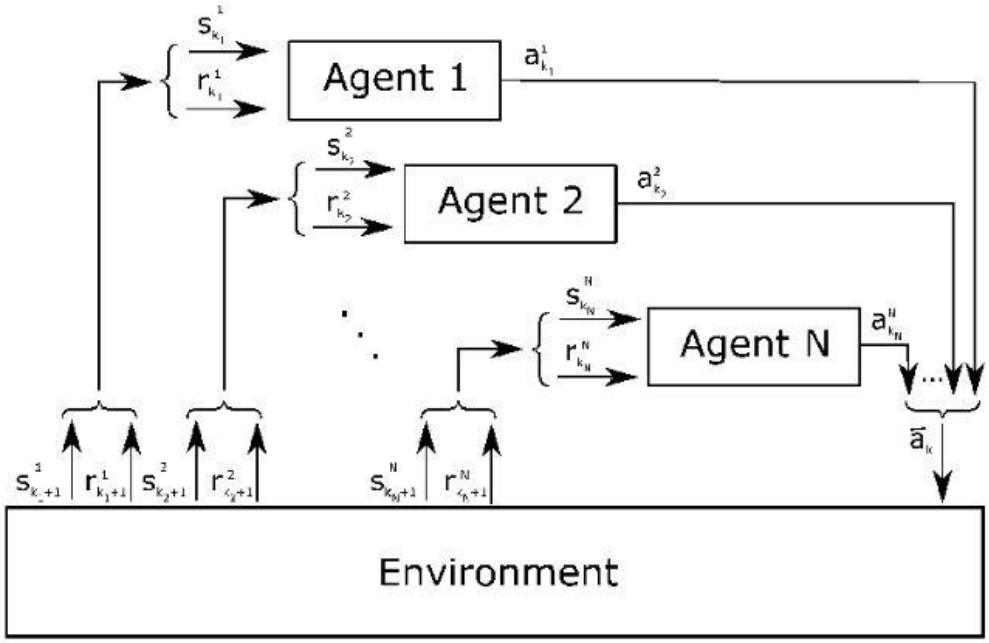


Figure 3.6: Multi Agent RL [14]

3.12 Types of Agents in a MARL System

According to the behavior of agents, the agents in a MARL system can be categorized in to three types as Co-operative, Competitive and Mixed [16],[23],[32],[33],[35],[38],[40].

- **Cooperative**: The same reward is earned by all agents and all agents work together to get the reward. This is essentially perceived to be a team incentive.

- **Competitive**: In order to get a reward, each agent (or group of agents) competes. The incentive cannot be shared between agents or agent communities.

- **Mixture of both (Mixed)**: This kind of agent is a hybrid of both; it can be competitive as well as cooperative. In a game like basketball, for instance, where play is cooperative between teammates and competitive between two teams.

3.14 Summary

This chapter offers an in-depth overview of the technological background and scientific implications of both reinforcement learning and multi-agent schemes. Next chapter explains the Approach of the research with key components of the approach.

Chapter 04

Approach

4.1 Introduction

In chapter 03, the technology adopted in the solution to dynamic task scheduling in an unforeseen environment using multi agent reinforcement learning and enhanced Q-learning was discussed. This chapter presents the approach engaged by the researcher in driving towards a solution.

4.2 Hypothesis

The hypothesis of the research is that Multi Agent Reinforcement Learning and Enhanced Q learning can be adopted to solve the problem of dynamic task scheduling in an unforeseen environment. Behind this hypothesis, inspiration comes from how tasks in a honeybee or ant colony are assigned and how they are ensuring effective consistency and balance in day-to-day routines.

4.3 Input

The Inputs for the system would be the input heuristics in terms of environmental conditions like number of agents involved, current state of the environment and backlog of tasks and priority.

4.4 Output

The output would be set of scheduled tasks for the dynamic work environment.

4.5 Process

The process followed or the technology adopted would be Multi Agent Reinforcement Learning and Enhanced Q-Learning.

First, the environmental conditions like number of agents involved, current state of the environment and backlog tasks will be inputted to the scheduler. From the enhanced Q learning algorithm embedded in the Scheduler, the value function or the policy function will be outputted. It will be presented to the environment, and environment will decide the amount of reward to be gain by the agent at each time. In each training episode it will try to maximize the reward it gets from the environment. Agents will communicate, negotiate and coordinate the

tasks themselves by the Multi agent reinforcement learning (MARL) component. Thereby, the backlog tasks will be scheduled for each of the agents. The state that results the maximum rewards would be chosen as the optimal scheduled state of the environment. Instead of reacting with the real environment continuously, a simulated model-based environment would be generated initially and it would react with real environment only after certain number of episodes in the simulation run. The process flow can be illustrated as in below flowchart in Figure 4.1.

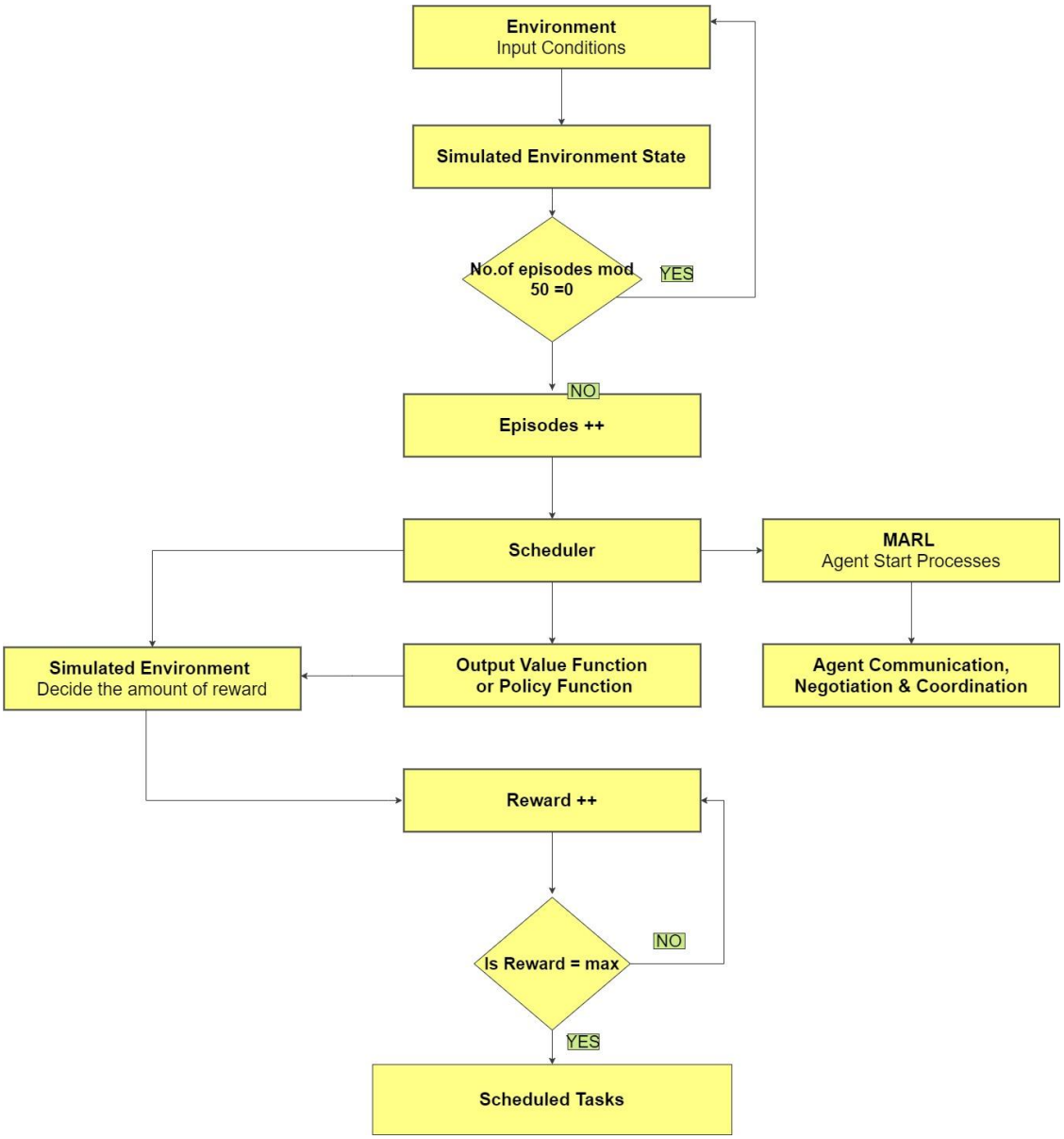


Figure 4.1: Proposed System Flowchart

4.6 Features

The overall features of the system include the following.

- Model-free and model-based RL integration
- Standalone simulation application
- Optimized dynamic scheduling of tasks
- Priority based task scheduling
- Usability and can be customized
- Accessibility and easy to use
- Uncertain environment modelling (Can be modified/customized and adapt to the entirely new environments)
- Support for major agent behaviors

4.7 Users

The end users of this system would be researchers of reinforcement learning, project managers, system planners, project planners etc.

4.8 Summary

The hypothesis, input, output, features, users and process of the proposed solution were discussed in this section. The researcher clarified the input processing relevant to the proposed Reinforcement learning model. The method proposed and the procedures used to convert inputs to outputs have been defined. At the end of the approach chapter, the overall features of the new system were also addressed. The next chapter will introduce the concepts of the suggested solution.

Chapter 05

Design

5.1 Introduction

By designing the overall system design, system model, system architecture, software architecture, modular architecture, and simulation design, the design section explains the design of the system created. Using diagrams, the modules of the system and their relations have been developed. And what each module does in the framework has been addressed here.

5.2 System Design

The inputs for the system involve the Number of agents involved, current state of the environment and backlog of tasks and sub-tasks, Rewarding criteria and Number of steps of each episode. The outputs are a set of scheduled tasks for the work environment. The process going to follow in converting the inputs to outputs are Multi Agent Reinforcement Learning and Enhanced Q-Learning. Enhanced Q-Learning includes Deep Dyna-Q+ algorithm which is proposed for the purpose of dynamic task scheduling in an uncertain environment [41].

5.3 System Model

The system model with the agents, product backlog and tasks, rewards, time, environment, actions and state are designed as follows.

5.3.1 Agents

Agents are the individual machines which are responsible for performing the tasks. The individual machines have different capabilities and can act as agents that perform specialized tasks. The agents collectively work to complete the set of tasks in the backlog. They communicate with other agents to coordinate and negotiate the tasks. Some of the special tasks can be taken as direct subtasks toward the respected machine agents by the agents and handling as well as coordinating tasks when the machines are busy.

5.3.2 Product Backlog and Tasks

Product backlog includes the tasks that must be scheduled for the respective agent within the given simulation time through utilizing the allocated resources. This consists of the tasks, subtasks, their task priority and a measure of the number of targets achieved by the scheduler in the respective time period.

5.3.3 Rewards

The scheduler gets rewards for completion of tasks. In this kind of scenarios, Rewards can be gained in 2 different methods. One method is accumulating rewards once completion of sub-tasks. Second method is achieving rewards in completion of all sub-tasks and thereby completing a full major task. In this development, the second option is used and rewards has been taken once all the subtasks have been finished. Thus, the system state that incurs the maximum cumulative reward should be chosen by the scheduler as the optimal scheduled state of the environment.

5.3.4 Time

The total time is the time taken by a task to complete the whole process by achieving all the sub tasks. Apart from total time factor, the difference between current timestamp and previous timestamp before ‘n’ episodes are also taken into consideration in the proposed method.

5.3.5 Environment

The environment includes tasks and agents. In development, a simulated environment is generated to represent different tasks and agents. The scheduler interacts with the actual real environment and also with the generated simulated environment.

5.3.6 Actions

Actions in the Reinforcement learning model refers to the tasks that should be completed by the agents. The number of actions is as same as the number of goals the agents need to achieve within the simulation.

5.3.7 State

The environment state is subjected to change once the tasks are performing by the agents. The state should reach optimal scheduled state once all the tasks are achieved.

5.4 System Architecture

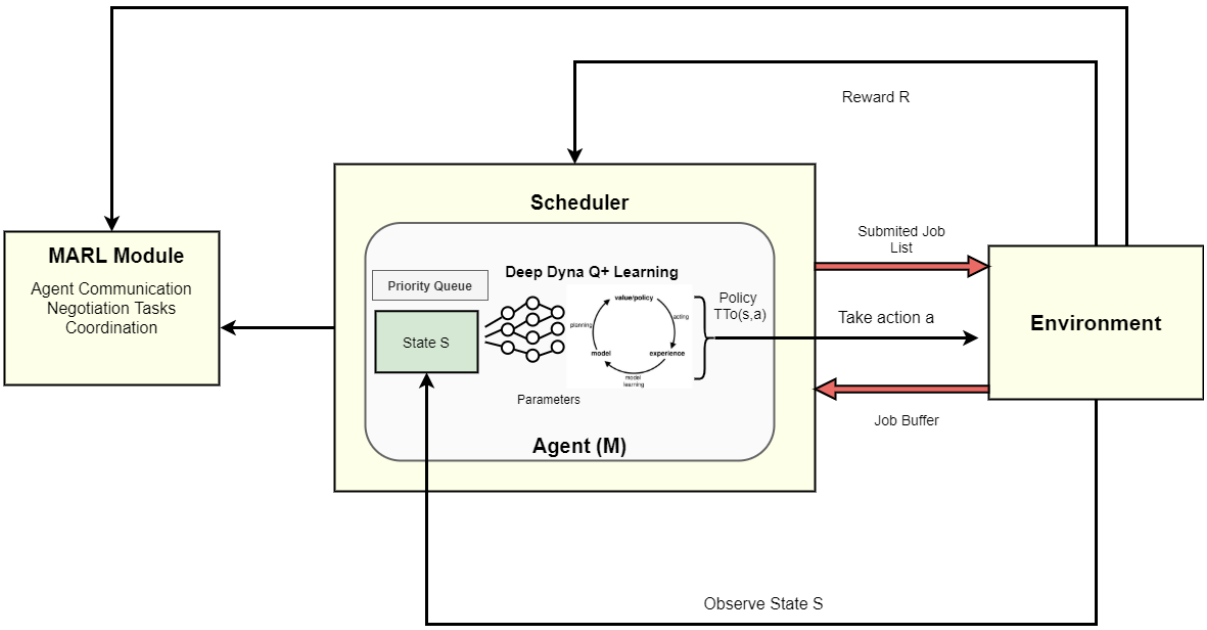


Figure 5.1: System Architecture.

The proposed solution includes 3 major components interconnected as shown in the Figure 5.1. The major components consist MARL module, Scheduler and the Environment. Each of the components interact with other major components as shown in above Figure 5.1.

5.5 Modular Architecture

The proposed solution is viewed as a modular design as proposed in below figure. It consists of major three tiers as Presentation Layer, Application Layer and Database Layer. The researcher identified major five modules that operate within these layers as, Data acquisition module, MARL module, Scheduler module, Environment module and Result logging module. The five major modules include five sub modules as categorized in the Figure 5.2. The design of the modules is being discussed in the next sub sections.

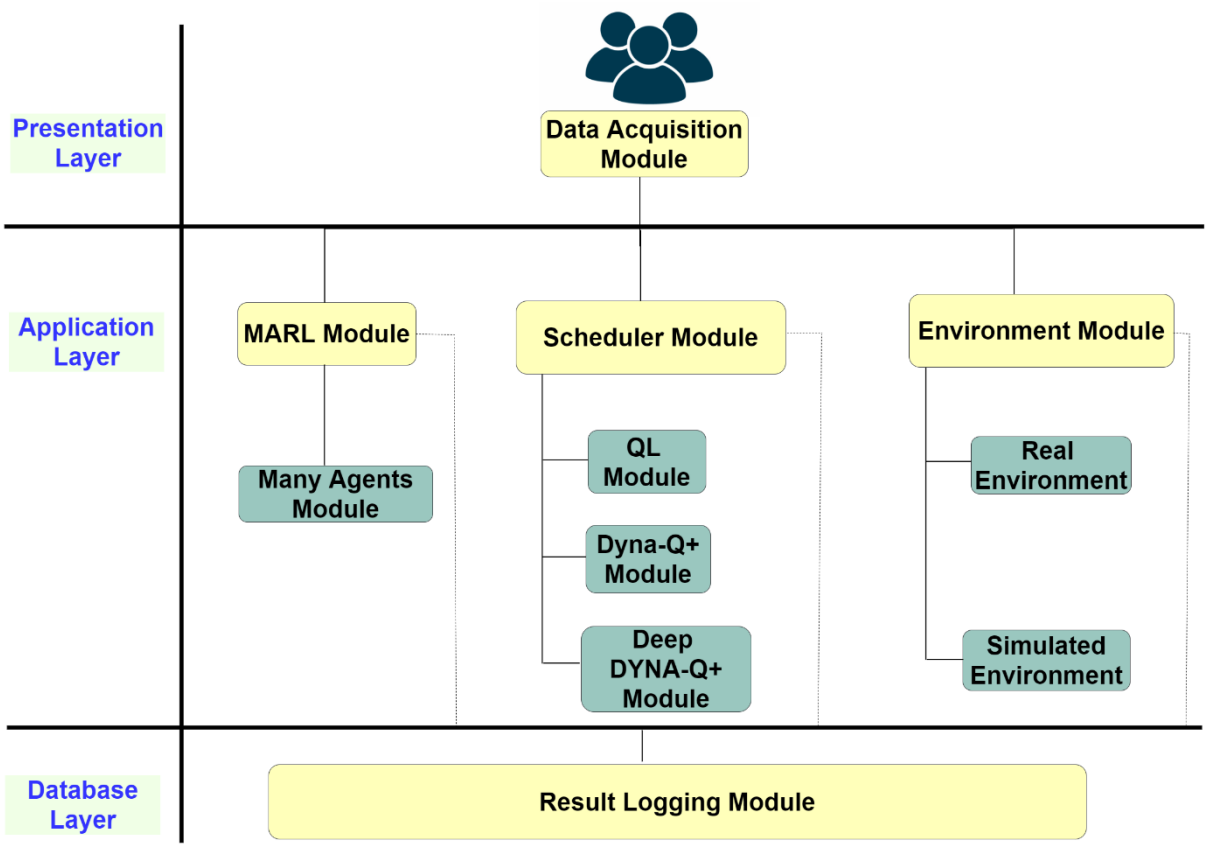


Figure 5.2: Modular Architecture

5.5.1 Data Acquisition Module

In Data Acquisition Module, the metadata on number of agents involved, current state of the environment, backlog of tasks and sub-tasks, rewarding criteria, priority of tasks, number of steps and the scheduling behavior would be determined. The proposed hybrid approach would follow online reinforcement learning and thus would accept real time data inputted by the user.

5.5.2 MARL Module

MARL module includes the machine agents involved with the scheduler. Multi Agent Reinforcement Learning concept is used in designing MARL module. Here the machine agents work together, and they are responsible in handling three major agent based functions.

i. Agent communication
ii. Agent negotiation
iii. Agent coordination

When the agents complete the tasks, they get rewards in return. Here the agents are designed as co-operative agents, that means the cumulative rewards gained by all agents is summed up. Both agents share the same reward earned and all agents collaborate to get the reward. This is

essentially called a team award won by all the agents of the participating system. MARL module interacts with both Scheduler and Environment modules.

5.5.3 Scheduler Module

Scheduler Module will determine the best planning schedule adhering to the priority queue, for the agents based upon the scheduling behavior specified in the Data acquisition module. The scheduling behaviors developed are of major two types. Q-learning is developed as a model-free approach and Dyna-Q+ learning and Deep Dyna-Q+ learning which has been proposed as the novel model-based approach. The model-free approach acts with the real environment specified by the user and deals with direct experience at each time step to learn. The model-based approach focuses on explicitly analyzing the value functions from environmental interaction.

i. Deep Dyna – Q+ Scheduler

In Deep Dyna Q+ learning, the Q-value function is calculated using a Deep Q Network (DQN). DQN takes state characteristics as raw data and outputs the value of the next state-action pair's Q function value [3, 33]. The loss function of the neural network is expected to achieve as designed in (1). The action values output by the DQN is inputted for the Dyna Q+ model developed.

$$[(r+\gamma max_a' Q(s', a; \Theta_1\text{-})\text{-}Q(s,a; \Theta_1))^2] \quad (1)$$

where, $(r+\gamma max_a' Q(s', a; \Theta_1\text{-})$ is the output of the Target network and $Q(s,a; \Theta_1))^2$ is the output of the Prediction network.

ii. Dyna – Q+ Scheduler

A reduced number of experiences with the real world is used by the structured model-based approach. The goal we have tried to achieve through a model-based approach is to create a model based on initial encounters with the environment and then use the expected environment resulting from previous modules to simulate future episodes. Dyna Q+ implementation, which is initially proposed by Sutton and Barto [14] has been developed in the simulation in order to cater the requirement of responding to unforeseen states. Two potential tricks were designed to keep the agents capable of exploring new states to suit the evolving world so that it pushes the agent to explore and offer reward. Via Dyna-Q+ implementation, one is proposed. The principle is completely the same as Dyna-Q, except that it keeps track of the amount of times a state has been visited and rewards a state that has not been visited for a long time. (since the state could have possibly changed as time goes on). The reward function is modified as in (2), where r is the modelled reward for a transition, and the transition has not been tried in τ time steps for some small κ. Dyna-Q+ development is more likely to detect the changing environment and respond positively by giving additional rewards to un-explored states. Another trick is to only adjust the Q table if the state condition in the previous setting does not exist.

$$r+ = r + \kappa * sqrt(\tau) \qquad (2)$$

iii. QL Scheduler

QL Scheduler includes the model-free design. The bellman equation defined in the methodology to find the next best action and its explanation is defined as in (3).

$$New\ Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma\ maxQ'(s',a') - Q(s,a)] \qquad (3)$$

where, New *Q(s,a)* is the new Q value for the state and action, *Q(s,a)* is the current Q value, *R(s,a)* is the reward for taking an action in a state, *maxQ'(s',a')* is the maximum expected future reward, α is the learning rate and γ being the discount rate.

5.5.4 Environment Module

The environment was setup as two sub-modules covering real environment and simulated environment. The simulated environment is generated based upon the initial environment and agent-environment-actions interactions within the episodes. The simulated environment interacts with the real environment only once in specified number of episodes defined, here we took it as once in 50 episodes. The environment is subjected to change dynamically which means it is unpredictable and uncertain. The machine agents dynamically interact with the real and simulated environments while the simulation is run.

5.5.5 Result Logging Module

In this module, the optimized optimal task schedule would be retrieved and saved in a suitable manner that enables reloading when necessary.

5.6 Simulation Design

In the simulation design sub-section, the modules designed in the System Design phase are integrated in an identified manner that is revisited in simulation implementation. The environment module consists of the global environment and the agents' local environments. The global environment represents MARL Module that includes the agents, their local environments and scheduler module. Apart from that global environment also include tasks, subtasks, priority of tasks, rewards and global state. The agents themselves have their agent state, local environment and the scheduler in each of them. The scheduler component each of the agents have supports agent learning and schedule tasks themselves. The summary of simulation design can be illustrated through Figure 5.3.

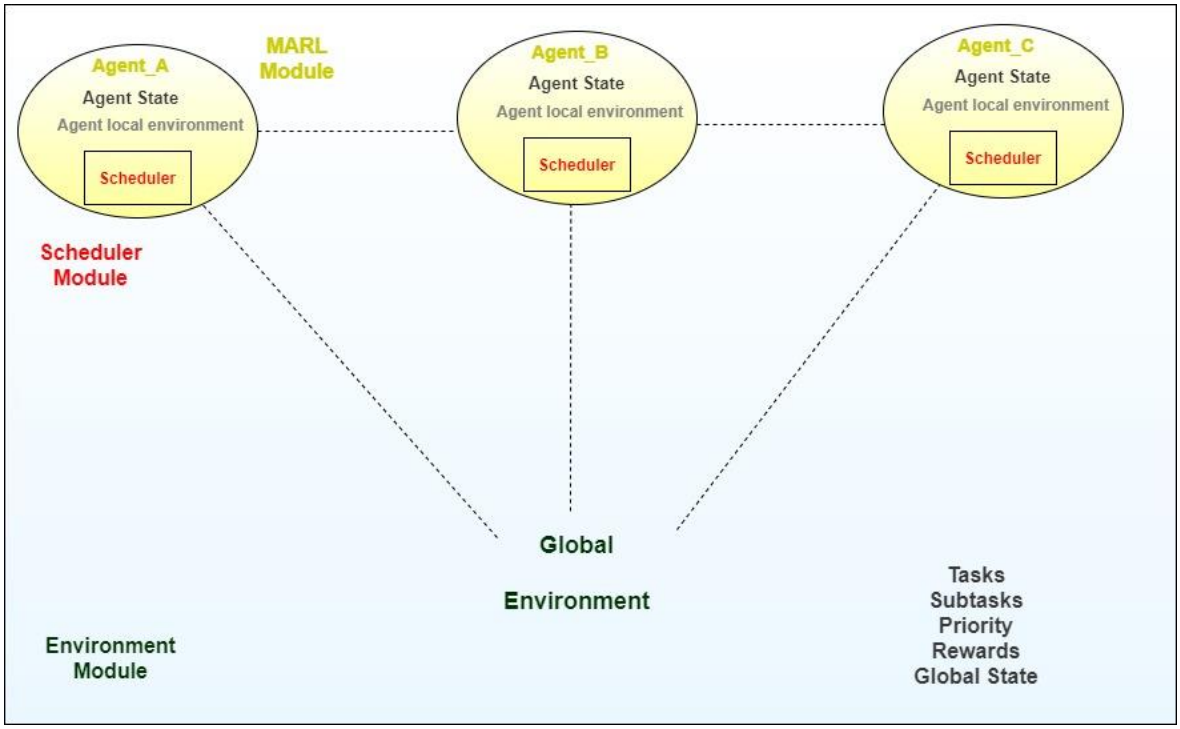


Figure 5.3: Simulation Design

5.7 Summary

The design process of the proposed system has been presented in this chapter. The overall system design, system model, system architecture, software architecture, modular architecture and simulation design were discussed here in the first place. Then, along with diagrams, all the defined modules with their interconnections and functions were created. In the next chapter we discuss about the System Implementation.

Chapter 06

Implementation

6.1 Introduction

The architectural design of the proposed framework was discussed in chapter 05. The implementation of the framework is presented in Chapter 06. Firstly, this chapter presents details on how development of the simulation was done and how each of the module were developed. Then, it is followed by the tools and technologies adopted in the implementation.

6.2 Implementation of Modules

6.2.1 Data Acquisition Module

In the data acquisition module, the metadata with respect to below parameters are obtained from the user.

- Number of Agents Involved
- Current State of the Environment
- Backlog of Tasks and Sub-Tasks
- Rewarding Criteria
- Number of Steps
- Priority of tasks

The scheduling behavior is also needed to be chosen by the user. The behaviors can be of major 2 forms that have 4 categories as below.

1) Model-Free Behavior- Random Learning Scheduler and Q-Learning Scheduler

2) Model-Free and Model-Based Behavior Integration- Dyna-Q+ Scheduler and Deep Dyna Q+ Scheduler

Online reinforcement learning concept is used in data acquisition; hence the real time input data is collected from the user from the Data Acquisition Window implemented shown in Figure 6.1.

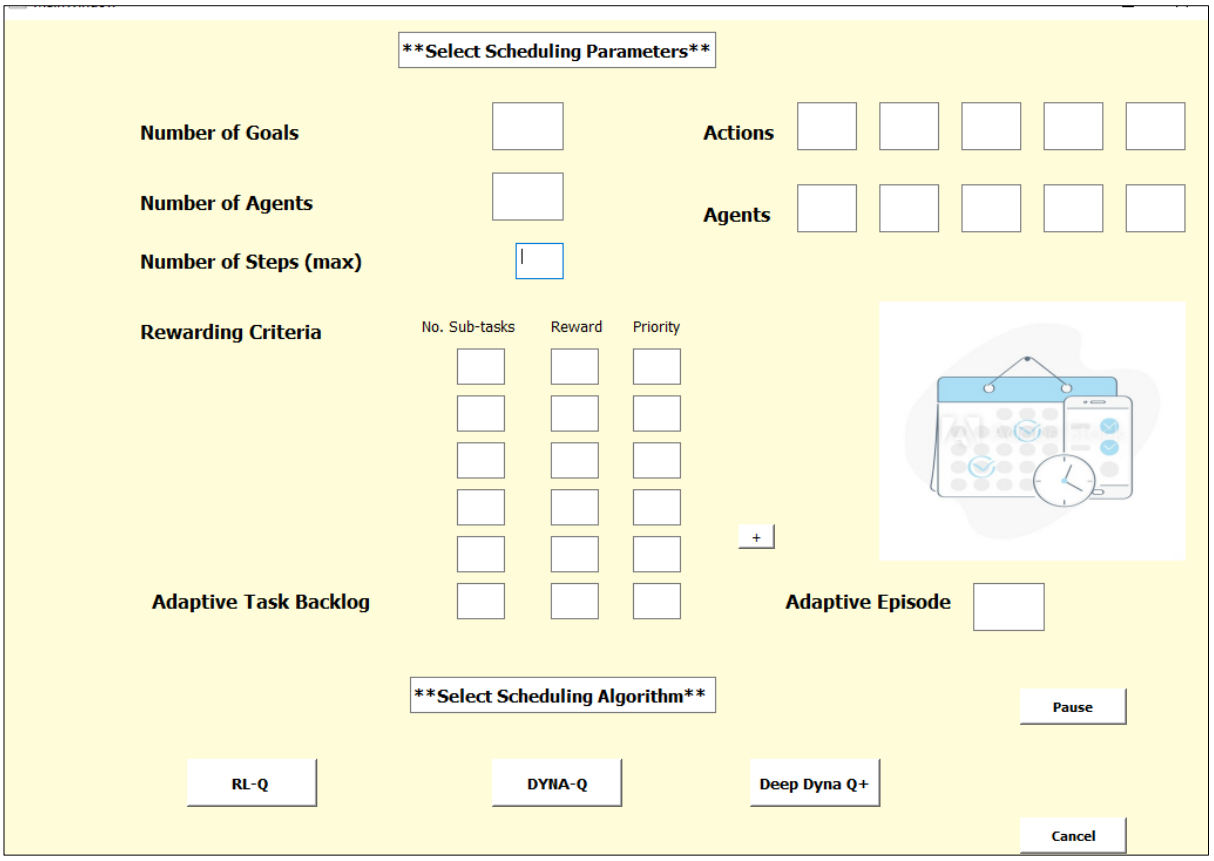


Figure 6.1: Data Acquisition Window

6.2.2 Environment Module

In this module, the environment is generated based on the criteria specified by user. The environment was setup as two sub-modules covering real environment and simulated environment. The local environment is generated based on the criteria specified by a user. The heuristics in terms of the number of machine agents, tasks, subtasks, priority, rewards, initial state and number of targets constitute the task backlog. Figure 6.2 shows a real sample environment generated and Figure 6.3 shows the task backlog that's developed.

Agents	Sub Tasks					Rewards
	1	2	3	4	5	
M1						
M2						
State Init=0,0						

Figure 6.2: Sample Environment Generated (for 3 tasks, 2 agents)

Task	Sub Tasks	Rewards	Target	Priority
Task 1 (T1)	5	3	1	10
Task 2 (T2)	4	10	2	9
Task 3 (T3)	3	5	3	1
	Total Sub Tasks-12	Total Reward-18	Total goals achieved-3	

Figure 6.3: Sample Task Backlog

6.2.3 Scheduler Module

Scheduler Module determines the best planning schedule adhering to the priority-based queue, for the agents based upon the scheduling behavior specified in the Data Acquisition Module. This involves three sub modules; QL Module, Dyna-Q+ Module and Deep Dyna-Q+ Module. As defined in this section, each of the modules will be implemented. As introduced, the reward function is suggested and enforced as follows.

6.2.3.1 Reward Function

The reward function explains how the rewards are being generated in the scheduler component. The cumulative reward is generated by reward obtained by the task completion and Dyna-Q+ reward. Reward from task completion is achieved by completing all the sub-tasks and thereby completing the respective main tasks by the involvement of several agents cooperatively. Dyna-Q+ reward is calculated at each episode for the unforeseen environment states by the equation κ*sqrt(τ), where r is the modelled reward for a transition, and the transition has not been tried in τ time steps for some small κ. Dyna-Q+ development is more likely to detect the changing environment and react positively by offering additional rewards to un-explored states. Potential reward shaping technique is done in order to shape the reward accumulation and it empowered smooth convergence. The implemented reward function is as in Figure 6.4.

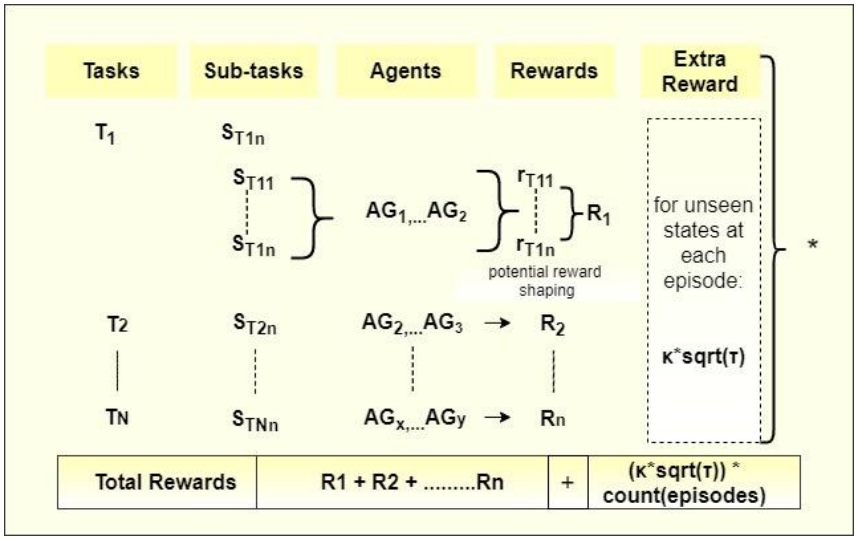


Figure 6.4: Reward Function Implemented

6.2.3.2 QL Module

Following pseudo code is proposed in this scheduling behavior. Figure 6.5 is a simple illustration on the Q-learning process adopted.

Pseudo-code for QL Module

1. *Initializing the local environment.*
2. *Start the learning process.*
3. *Select actions.*
4. *Update the environment and states.*
5. *Reinforcement learning by the scheduler.*
6. *Save the simulated RL model.*

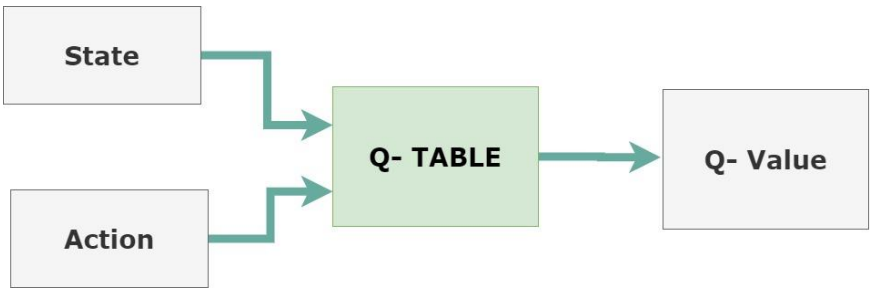


Figure 6.5: RL Q-Learning Process

6.2.3.3 Dyna Q+ Module

Following pseudo code is proposed in the implementation of the scheduling behavior. The maximum expected future reward is calculated using (4), where *self.timeWeight* corresponds to κ in (3) and the timestamp difference between current and previous timestamp of 'n' episodes defined by user.

Pseudo-code for Dyna-Q+ Module

1. Initializing the local environment.
2. Start the learning process.
3. Select actions as per task priority.
4. Update the environment and states.
5. Reinforcement learning using Dyna-Q+ learning by the scheduler
6. Save simulated RL model.
7. Simulate the environment and run n times
   for the RL to learn.
8. Capture the optimal scheduled state.

max_reward+= self.timeWeight * np.sqrt(self.time - _time) (4)

6.2.3.4 Deep Dyna-Q+ Module

Following pseudo code is proposed in this scheduling behavior. The sub section is followed by presenting the structure of Deep Dyna-Q+ learning model developed.

Pseudo-code for Deep Dyna-Q+ Module

1. Initializing the local environment.
2. Start the learning process.
3. States are the input to the Target Network.
4. Store all the past experience in the replay memory
5. Function approximation by the scheduler. The next action is determined by the maximum output of the Q-network (Evaluation Network)
6. Calculate the loss function. It is the mean squared error of the predicted. Used RMSProp optimizer here. Q-value and the target Q-value – Q*. (Bellman equation)
7. Dyna Q+ learning and save simulated RL model.
8. Simulate the environment and run n times for the RL to learn.
9. Capture the optimal scheduled state.

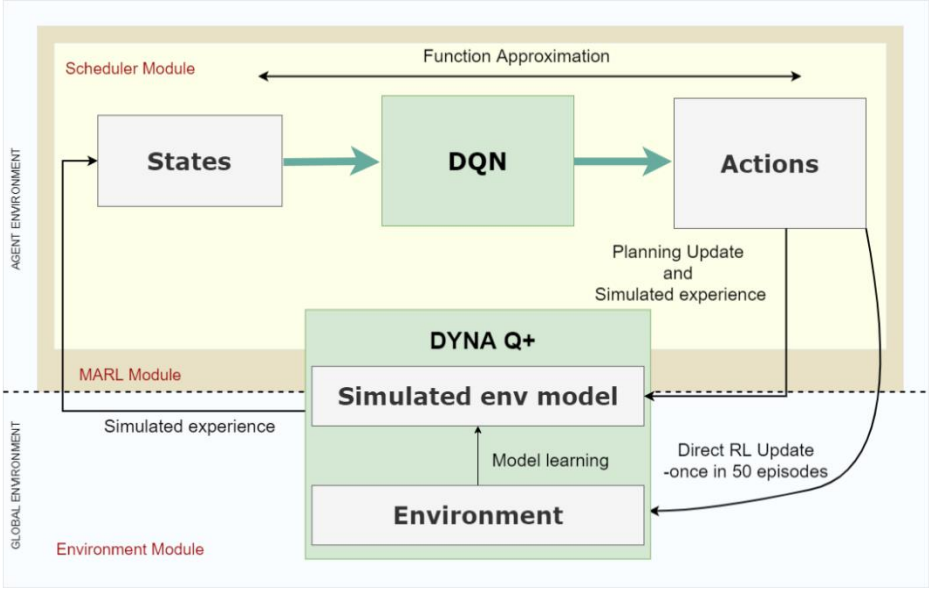


Figure 6.6: Structure of Deep Dyna-Q+ Learning Model Developed

Figure 6.6 presents the structure of Deep Dyna-Q+ learning model developed. Here, the Environment module is the major module that includes global environment and agent environment. The two sub modules of Environment module, simulated environment and real environment connects with both the global and agent environment as shown. Inside the Environment module, MARL module was developed. Scheduler module is implemented within

the MARL Module. The implementation setup is co-related with the simulation design introduced by Figure 5.3 in the Design chapter. The DQN implementation resides in the scheduler module. There, the environment states are inputted to the DQN and most appropriate actions are outputted through function approximation. The actions and planning update with simulated experience are inputted to the Dyna-Q+ module that resides in both the MARL module, Scheduler module and Environment module. The direct RL update is happening in between the Scheduler and real environment once in 50 episodes. Model training is done in Dyna-Q+ module and the simulated experience is rendered back to the Environment module in return as shown in the figure.

Designing and Training Deep Q-Network

The Deep Q-Network is implemented as Function Approximation of the action-value function q (s, a) of Dyna Q+ simulation. It includes several optimization techniques that are listed below.

Replay Memory: Since the machine agents acts in the real environment and explore the world, the neural network is not trained on the go. Here, the experiences of the agents were stored inside a buffer space and the model was trained by sampling out the batches of experiences using prioritized experiences.

Target Network and Evaluation Network: Here, DQN is implemented using two Q networks as Target network and Evaluation (Local) network each comprised of 2 layers as shown in Figure 6.8. The input of the DQN is the environmental states and the output is the estimated q-values of the state-action pairs as shown in Figure 6.7. The final aim of DQN is minimizing the loss function by gradient descent mechanism.

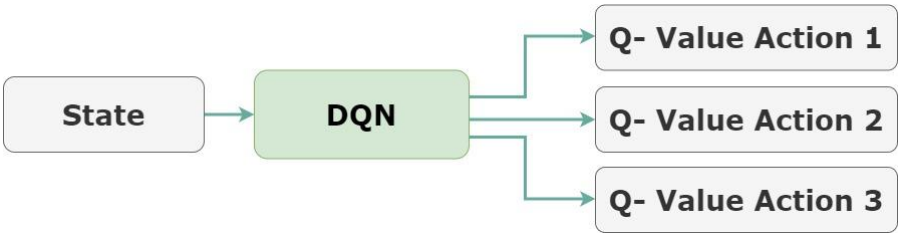


Figure 6.7: DQN Function Approximation

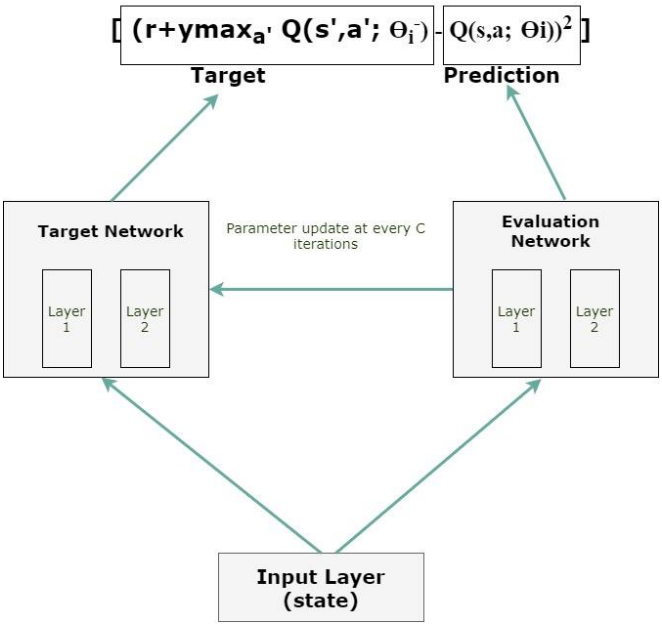


Figure 6.8: Structure of DQN Developed

As such, Deep Dyna-Q+ is able to handle realistic situations of observation space being too high-dimensional and large to be stored in Q table. Figure 6.9 shows the final scheduled environment that is expected to have generated from successful simulation run.

Agents	Sub Tasks							Rewards
	1	2	3	4	5	6	7	
M1	T1	T2 →	T2 →	T2 →	T1	0	0	3
M2	T2 ↗↘	T1 →	T1 →	T1 →	T3 →	T3 →	T3	10+5
State Init=0,0	T1-1,T2-1	T1-2,T2-2	T1-3,T2-3	T1-4,T2-4	T1-5,T2-0			18
Completed Tasks	0	0	0	1	2	2	3	

Priority queue based task allocation

Figure 6.9: Final Scheduled environment

6.2.4 MARL Module

Cooperate machine agents were utilized in the proposed framework. The individual machine agents can communicate using a simple messaging mechanism at each state when the simulation runs. The agents can coordinate the tasks between themselves and negotiate with other agents involved in the operation. The MARL concept is developed in the agent-based simulation. The simulation is premeditated such that the model reacts with the real environment once only 50 running episodes that dynamically adapts the learning based on the changing environment. The MARL setup embedded with the Dyna-Q+ and Deep Dyna-Q+ algorithm has been visualized in Figure 6.10.

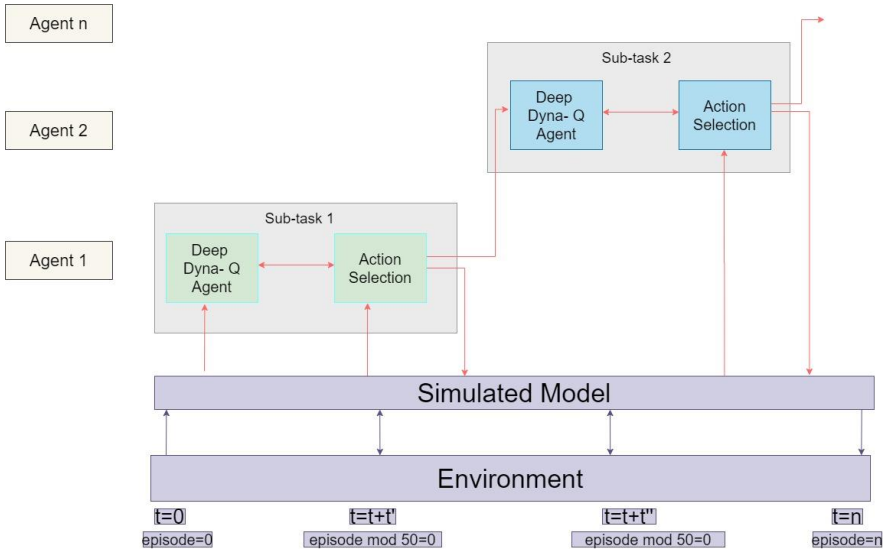


Figure 6.10: Agent Communication, Tasks Co-ordination

6.2.5 Result Logging Module

Save the schedule. Here, the simulation, Q table and logs in Deep Dyna-Q+ are saved locally. The final schedule is saved as a .jpg image.

6.3 Action Selection Mechanisms Developed

E-epsilon decay action selection mechanism is used in selecting the next best action in the simulation run. Here the epsilon is reduced, since we need less and less exploration and more exploitation while the agents interact with the environment. The exploration rate is defined by epsilon and the exponential decay rate for the exploration probability is defined by the decay_rate. Two constants max_epsilon and min_epsilon is used to define the boundary of exploration. The code logic implemented is as in (5).

self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon) * np.exp(-self.decay_rate * episode) (5)

E-epsilon decay action selection mechanism is much more suitable if the initial environment is pre-defined and is discrete throughout the simulation. But if the environment is continuous throughout the simulation, it was observed that E-epsilon greedy would not incur best results. Instead, Epsilon-greedy would perform considerable good results. In epsilon-greedy, always the action with the highest value is picked always. Here, a uniform random number is taken (delta), and if delta is less than a small constant value, (epsilon), then an action is picked randomly from the set of all available actions for a given state. The code logic implemented is as in (6).

self.epsilon = 0 if e_greedy_increment is not None else self.max_epsilon

self.epsilon = self.epsilon + self.epsilon_increment if self.epsilon < self.max_epsilon else self.max_epsilon (6)

6.4 Generalization to Unseen Environments

One of the main aim this research is focused is generalization to unseen environments. As a solution, the scheduling solution was implemented such that it dynamically responds to unseen environment conditions. It was implemented in two levels as System level and User level.

6.4.1 System Level

In System level, the generalizability is catered in three mechanisms. Two mechanisms are implemented through the Dyna-Q+ implementation. Here, the first principle is completely the same as Dyna-Q, except that it keeps track of the amount of time a state has been visited and rewards a state that has not been visited for a long period of time. (since the state could have possibly changed as time goes on). The reward function is updated as in (7), where r is the model reward for a transformation, and for some small κ, the transition was not attempted in τ time steps. Dyna-Q+ development is more likely to detect the changing environment and react positively by offering additional rewards to un-explored states.

$$r+ = r + \kappa*sqrt(\tau) \qquad (7)$$

It is coded as below in (8), a constant *self.timeweight* is used for κ and time difference is calculated for difference between 100 episodes where *_time* is the timestamp of previous 100^{th} episode and *self.time* is current episode. In this way the simulation can capture environment changes within a period of 100 episodes.

max_reward+= self.timeWeight * np.sqrt(self.time - _time) (8)

The second mechanism of generalization is, updating the Q table only if state is not existing in the previously visited environment.

The third mechanism is achieved through DQN part of Deep Dyna-Q+ implementation. In the DQN, the replay memory can store the transitions and that memory can be utilized in later times. By sampling transitions from the replay memory, the network can increase its ability to generalize.

6.4.2 User Level

In User level, the user can make a change in the environment by modifying tasks, subtasks, rewards, priority or add/remove them and thereby continue with the scheduling. Here, at each episode the Dyna-Q+ scheduling agent reacts with the real environment, an application delay is introduced in the execution of the program and enables space to make any changes.

6.5 Tools and Technologies

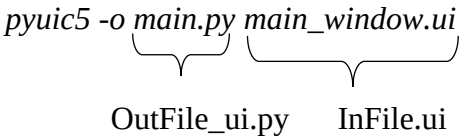
6.5.1 Integrated Development Environments (IDEs) Used

6.5.1.1 JetBrains-PyCharm (IDE)

PyCharm, specifically for the Python language, is an Integrated Development Environment (IDE) used in computer programming. PyCharm is a cross-platform framework that operates with versions of Windows, MacOS, and Linux. Some of the major features PyCharm that supported in this development are; Coding Assistance and analysis, Project and code navigation, Integrated python debugger, Python refactoring and Version control integration etc.

6.5.1.2 QtDesigner

Qt Designer was used to develop the system's User Interfaces. The installer of PyQt comes with the Qt Designer GUI builder tool. Without having to write the code, a GUI interface can be easily developed using its easy drag and drop interface. Qt Designer is not a Visual Studio-like IDE. Therefore, the facility to debug and develop the application is not available for Qt Designer. The designed layout is saved as main_window.ui in this tooling environment. The ui file includes an XML view of widgets and their design properties. By using the command line utility pyuic5, this design is converted into a Python equivalent. This utility is an uic module wrapper. The usage of pyuic5 is as follows –



Widgets and forms generated with Qt Designer are seamlessly incorporated with programmed code, which enables programmers to easily allocate actions to graphical elements using Qt's signals and slots mechanism. Within the code, all properties set in Qt Designer can be dynamically modified.

6.5.2 Languages Used

Python 3.5 is used as the main programming language for the application.

6.5.3 Libraries Used

6.5.3.1 Tensorflow

TensorFlow is an end-to-end open source platform initially developed by Google for advanced machine learning developments. TensorFlow is popular in its applications and usage in develop and train machine learning models. In our application, Tensorflow 2.0.0 is used in developing and training the Deep Q Network.

6.5.3.2 Tabulate

Tabulate is a library and a command-line utility used in representing the tabular data in the simulation.

6.5.3.3 OpenAIgym

OpenAigym or best known as Gym, is a toolkit for the implementation and comparison of reinforcement learning algorithms. The utils package of OpenAIGym 0.17.1 was used in developing the custom simulation environment.

6.5.3.4 Matplotlib

Matplotlib is the general Python programming language plotting library and is used in the proposed solution for visualizing purposes.

6.5.3.5 Plotly

Plotly is an interactive graphing library which was used in generating the task schedule and animating the schedule in visual form. *Plotly.express* edition was used in here. Some of interesting features of plotly such as; pan, box select, lasso select, zoom-in, zoom-out, auto scale, reset, download plot as png, toggle spike lines, show and compare data on hover increased the quality and visual capability of the final task schedule designed.

6.5.3.6 PYQt5

In developing the system's key user interface, PyQt was used. PyQt is a Qt application system collection of Python bindings and runs on all Qt enabled platforms, including Windows, OS X, Linux, iOS, and Android. PyQt is available in two variants, such as PyQt4 and PyQt5. Qt version 5 is provided by PyQt5. PyQt4 supports version 4 of Qt and will compile against version 5 of Qt.

6.5.4 Hardware Used

6.5.4.1 Graphics Processing Unit (GPU)

Intel® UHD Graphics 620 GPU is used as resource requirement.

6.6 Summary

This chapter presented the implementation process of the developed system. Here, the details on implementation of the modules and the tools and technologies adopted have been presented in detailed. The modular designs identified in the Design chapter were implemented by means of introducing software architectures and this chapter presented the implementation details in detailed. Next chapter will concentrate on the Evaluation of the developed system.

Chapter 07

Evaluation

7.1 Introduction

This chapter contains the Evaluation process which is used to evaluate and test the Reinforcement Learning framework which is discussed during this documentation. The evaluation process has been subdivided to several sections. The first section focus in testing the experimental design developed. The second section presents evaluation outcomes performed on the analysis of results based on the simulation.

7.2 Experimental Design

The experimental design for the evaluation purpose was premediated using selected scheduling scenario instances designed. The instances were setup considering following evaluation criteria.

- Varying agents performing tasks.
- Varying number of tasks, sub-tasks from less complex to high complex.
- Compare performance in Random Learning, Q-Learning, Dyna-Q+ and Deep Dyna-Q+ simulations.
- Varying rewarding criteria from less complex to high complex.
- Leveraging less prioritized and high prioritized tasks.
- Investigate performance between action selection mechanisms deployed.

Evaluation scenarios setup here focus to evaluate the models' performance with respect to above identified criteria. The instances will be elaborated in detailed in the next section. The simulation run results of Dyna Q+ Learning, Deep Dyna-Q+ Learning and the visualization of the task schedule in general, have been presented in this section.

7.2.1 Dyna Q+ Learning Simulation Results

The snapshots of the Dyna-Q+ simulation run has been provided under this section. Figure 7.1 presents the maze view of the representation of job completions. Figure 7.2 presents the state representation along with the job completions in the simulation run. Figure 7.3 illustrates the agent interaction, states and actions in an instance of the simulation run. Figure 7.4 is a sample Q-Table saved as a .csv file.

Figure 7.1: Representation of Job Completions (Maze view)

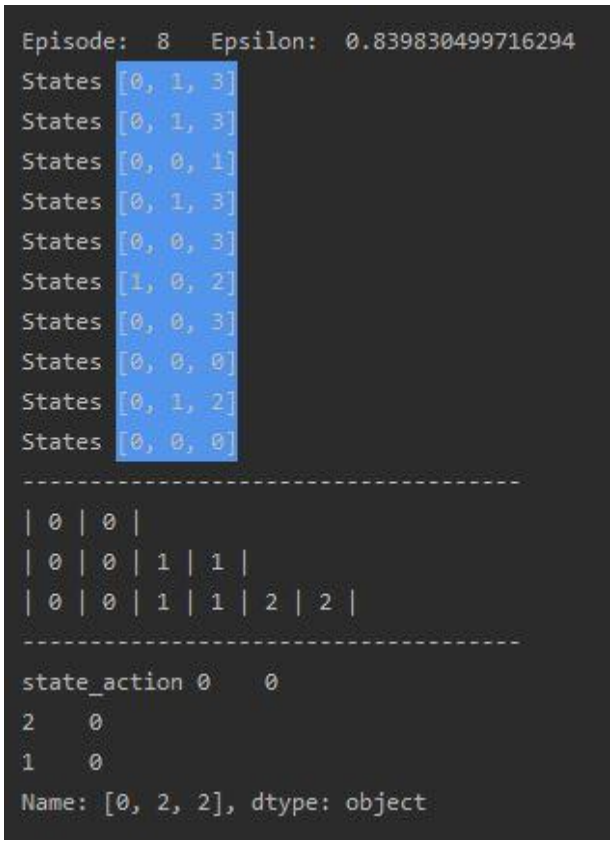


Figure 7.2: Representation of Job Completions

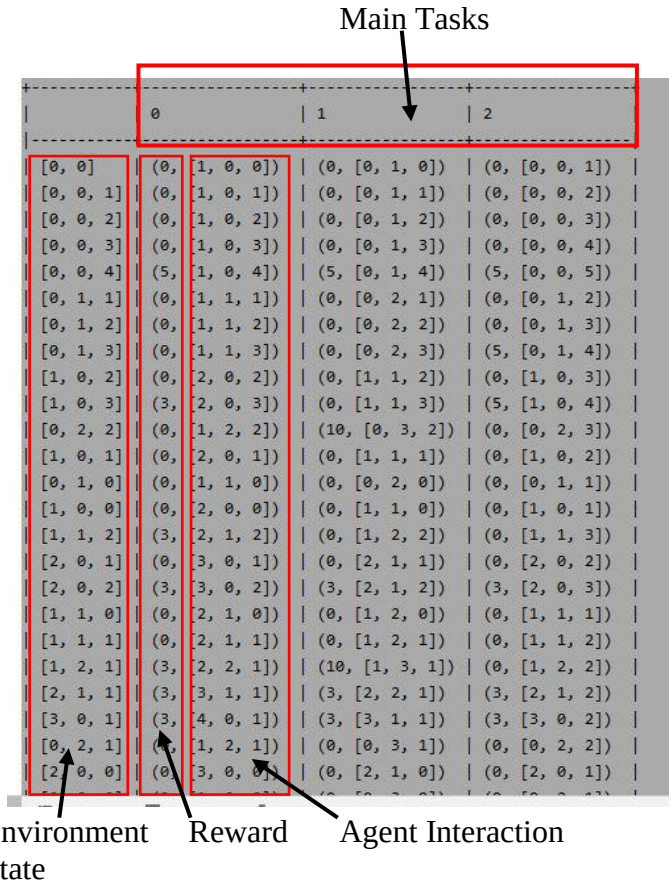
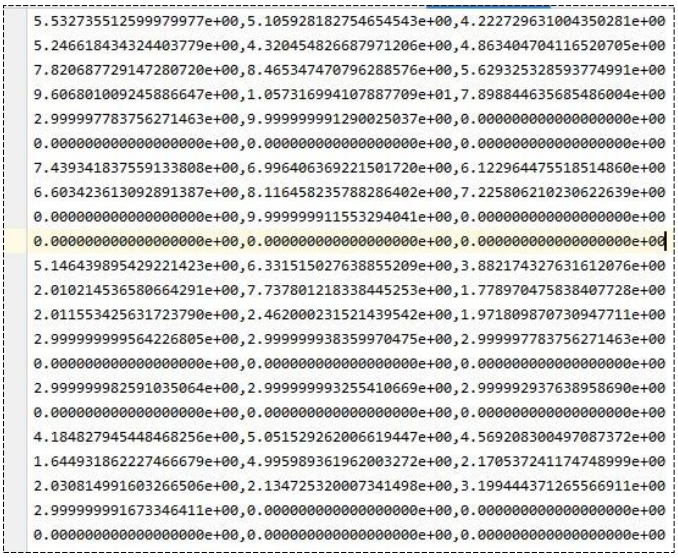


Figure 7.3: Agent Interaction, States and Actions

Figure 7.4: Saved Q-Table



7.2.2 Deep Dyna-Q+ Network Learning Simulation Results

The snapshots of the Deep Dyna-Q+ simulation run has been provided under this section. In the simulation run, the main tasks, current environment state, next predicted state, subtasks, agents, agent message passing, agent interactions and job completions are shown as in below Figure 7.5.

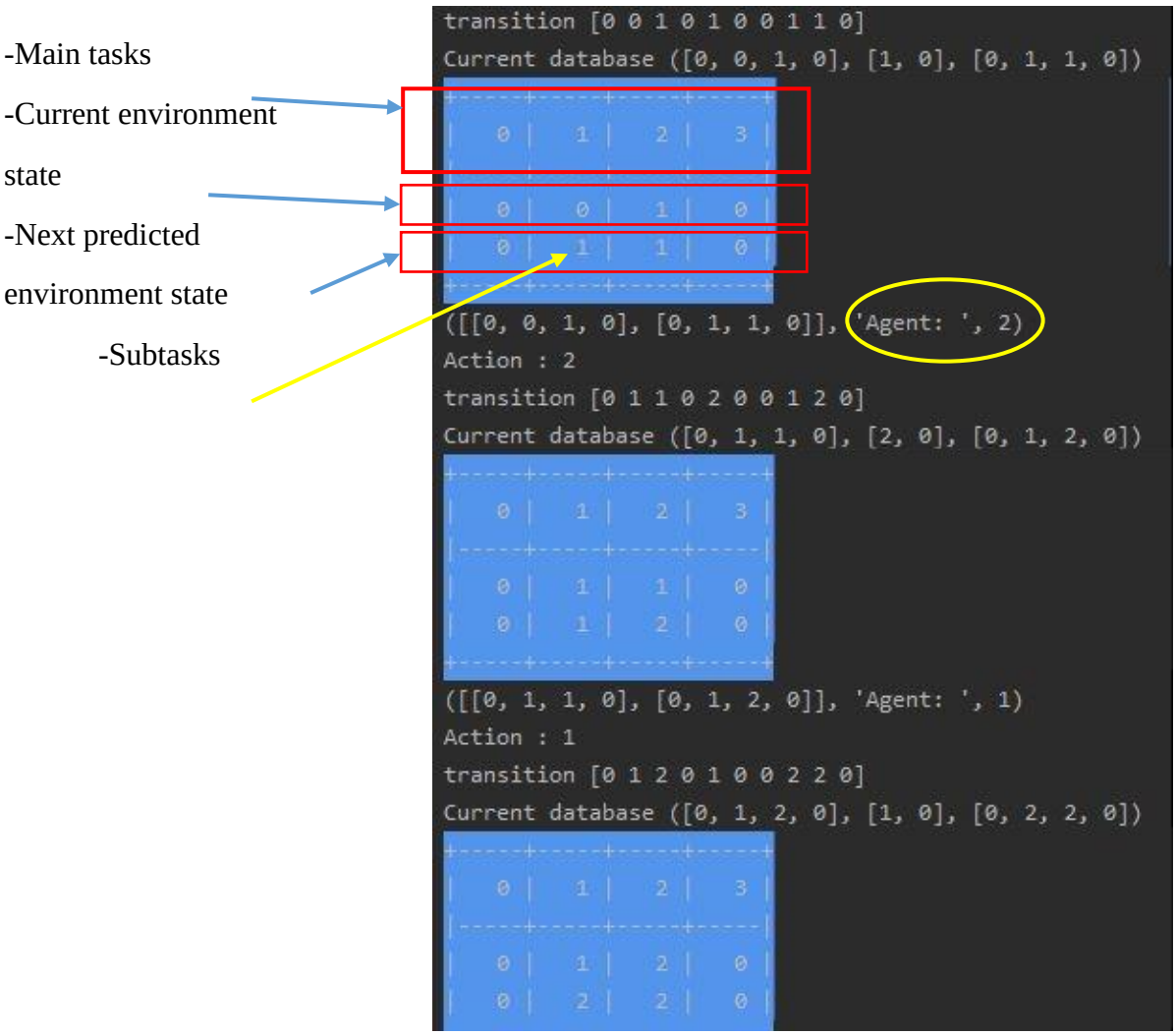


Figure 7.5: Agent Message Passing, States Representation and Job Completions in Deep Dyna-Q+

7.2.3 Visualization of Task Schedule

At the end of the simulation, the created task schedule is showed and saved as a plot as in Figure 7.6. The plot was developed from Python Plotly library. Some of interesting features of plotly including pan, box select, lasso select, zoom-in, zoom-out, auto scale, reset, download plot as .png, toggle spike lines, show and compare data on hover increased the quality and visual capability of the final task schedule designed. Some of the interactive options have been depicted in Figure 7.7 and Figure 7.8.

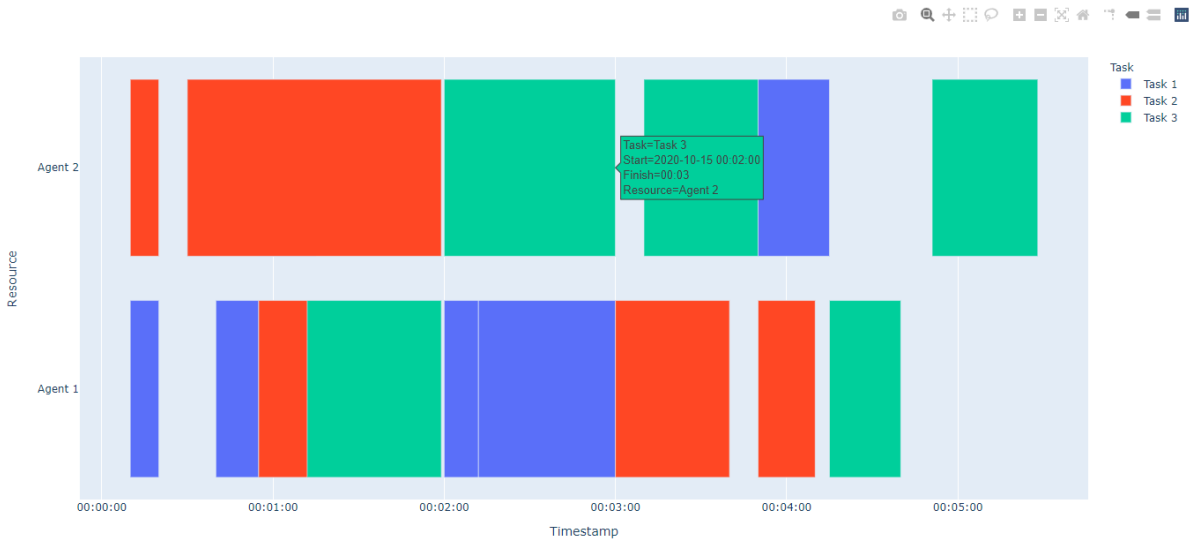


Figure 7.6: Sample Task Schedule Resulted from Deep Dyna-Q+ Simulation

Figure 7.7: Isolating Traces (such that flow of only needed tasks can be viewed)

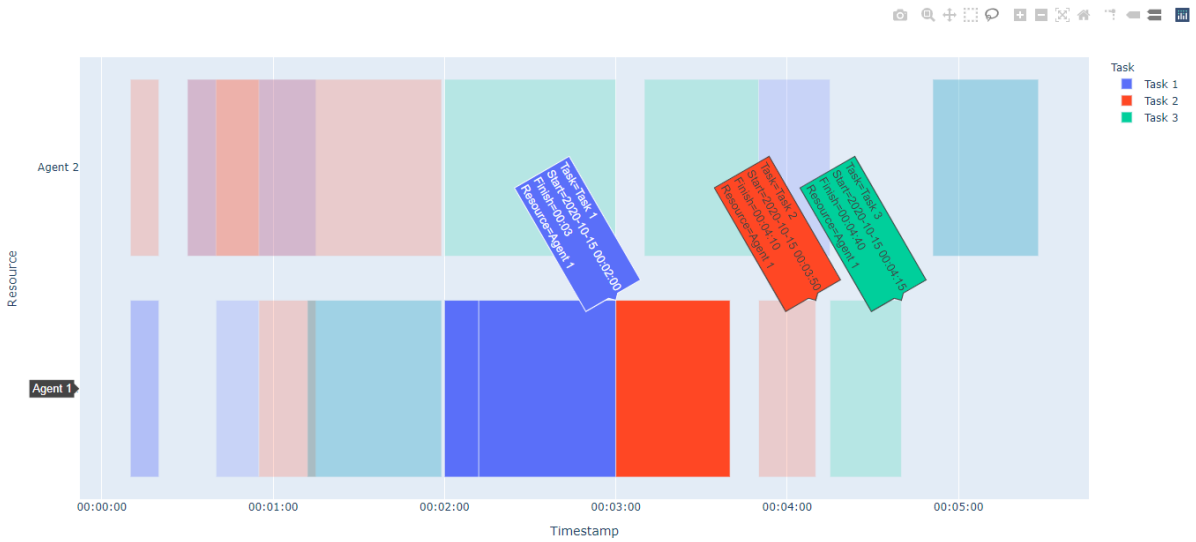


Figure 7.8: Lasso Select and Compare Data on Hover

7.3 Analysis of Results

Selected evaluation metrics are being used in order to validate the developed model-based and model-free simulation approaches. The metrics deal are as follows.

- Fine tuning the parameters.
- Performance metrics like makespan, throughput, response time.
- Selection of parameters.
- Hyper parameter tuning in Deep Q network.
- Variation of output with number of agents.

This section explains the evaluation of the proposed system in relation to the evaluation metrics using the scheduling scenario instances designed.

7.3.1 Reward Analysis

Here, the accumulation of rewards in each of the simulation run is analyzed. The evaluation results of one instance taken is shown below. Table 7.1 presents the sample task backlog defined and Figure 7.9 presents the reward plot results of random learning, QL simulation, Dyna-Q+ simulation and Deep Dyna-Q+ simulation respectively. As per the evaluation results, it was able to see that Deep Dyna-Q+ simulation can reach the maximum rewards accurately by completing all the tasks and subtasks. In Dyna-Q+ simulation, firstly it showed some improvement, but when tasks increase the reward was not converged and showed drastic variance. Both QL simulation and random learning simulation was not able to complete the tasks in this simulation instance.

Table 7.1: Task Backlog -Evaluation I

Task	Number of Agents	Sub Tasks	Rewards	Target	Priority
Task 1 (T1)	3	7	5	1	2
Task 2 (T2)	3	8	10	2	3
Task 3 (T3)	3	5	5	3	1
		Total Sub Tasks-20	Total Reward-20	Total goals achieved-3	

Figure 7.9: Plot of Reward Analysis I

7.3.2 Evaluation of the Throughput in Deep Dyna-Q+ learning.

Here, the variation in Throughput in Deep Dyna-Q+ simulation was tested against the Number of tasks and reward. Simulation was run by different real time random instances taken. The time was measured in timestamps. Results are as in Table 7.2.

Table 7.2: Results-Evaluation II

Expected Max_Reward	Actual Max_Reward	Throughput-Time (Timestamp)	Max_Reward Episode
28	27	103.963085	900
32	32	120.78	950
38	38	160.234	1010
43	42	221.3769	840
50	51	303.169	1120

7.3.3 Evaluation of Variation of output with number of agents in Deep Dyna-Q+ learning

Here, the reward variation in Deep Dyna-Q+ simulation was tested against the number of machine agents. Tasks, subtasks, priority were kept as constant. The task backlog considered is as Table 7.3.

Number of episodes run -2000

Table 7.3: Task Backlog -Evaluation III

Task	Sub Tasks	Rewards	Target	Priority
Task 1 (T1)	7	12	1	10
Task 2 (T2)	3	9	2	3
Task 3 (T3)	4	8	3	9
Task 4 (T4)	5	10	4	2
Task 5 (T5)	6	3	5	1
	Total Sub Tasks-25	Total Reward-42	Total goals achieved-5	

The observed results as per the maximum reward reached and the episode where the maximum reward is gained against the number of agents is as in Table 7.4. In this experiment, it was seen that the Deep Dyna-Q+ simulation achieves best results (that is closer reward to the maximum expected reward) when four machine of agents are used.

Table 7.4: Results-Evaluation III

Number of Agents	Max_Reward	Max_Reward Episode
2	47.28	840
3	47.27	840
4	43	850
5	29.6	1500
6	29.64	1500
1	29.64	1500

7.3.4 Evaluation of Variation of output with performance between Random learning, Reinforcement learning, DQ learning and Dyna-Q+ learning

The reward variation in between the four approaches; Random learning, QL learning, Dyna-Q+ learning and Deep Dyna-Q+ learning simulation was tested against the number of machine agents. Tasks, subtasks, priority were kept as constant. The task backlog considered is as Table 7.3. As per the evaluation results, it was seen that Deep Dyna-Q+ simulation was able to achieve maximum reward while completing all the tasks in a lesser number of episodes.

Number of episodes run -2000

Number of agents taken - 3

Table 7.5: Task Backlog -Evaluation IV

Task	Sub Tasks	Rewards	Target	Priority
Task 1 (T1)	7	12	1	10
Task 2 (T2)	3	9	2	3
Task 3 (T3)	4	8	3	9
Task 4 (T4)	5	10	4	2
Task 5 (T5)	6	3	5	1
	Total Sub Tasks- 25	Total Reward- 42	Total goals achieved- 5	

Table 7.6: Results- Evaluation IV

Learning Mechanism	Max_Reward	Max_Reward Episode
Random learning	6	1100
Reinforcement learning	20	2850
Dyna-Q+ learning	29.6	1500
Deep Dyna-Q+ learning	43	850

7.3.5 Cost Analysis

In cost analysis, the cost incurred in the learning process is estimated via number of episodes. Cost was analyzed for Dyna-Q+ and Deep Dyna-Q+ simulations. Once running the Dyna-Q+ simulation, the cost seemed gradually decreasing when the agents are simulated in the environment. Figure 7.11 presents the sample cost deviation result of an instance tested via Dyna-Q+ simulation.

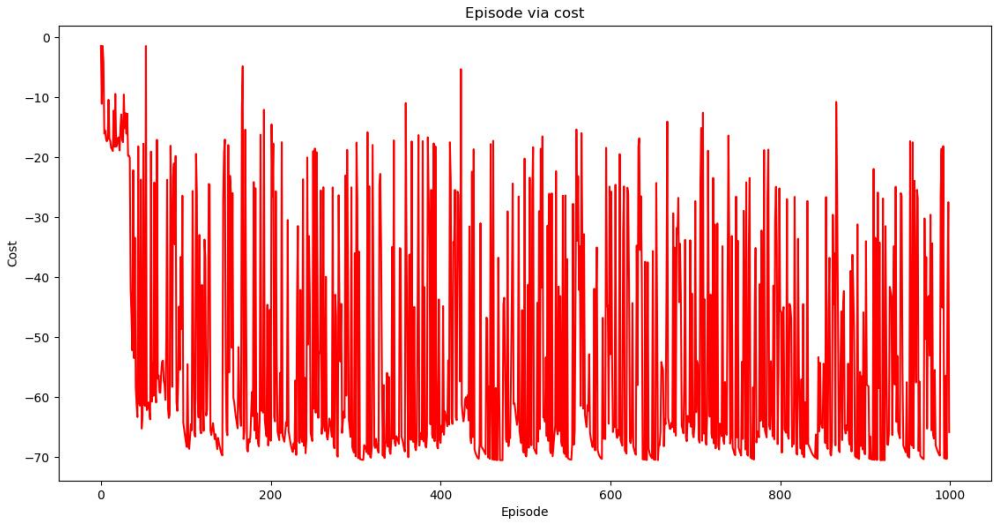


Figure 7.11: Cost Analysis of an instance in Dyna-Q+ Simulation

When running the same in Deep Dyna-Q+ simulation, the cost seemed gradually increase on RL learning and after the environment enter the scheduled state (state with maximum reward), the cost gradually decreased. Figure 7.12 presents the sample cost deviation result of an instance tested via Dyna-Q+ simulation.

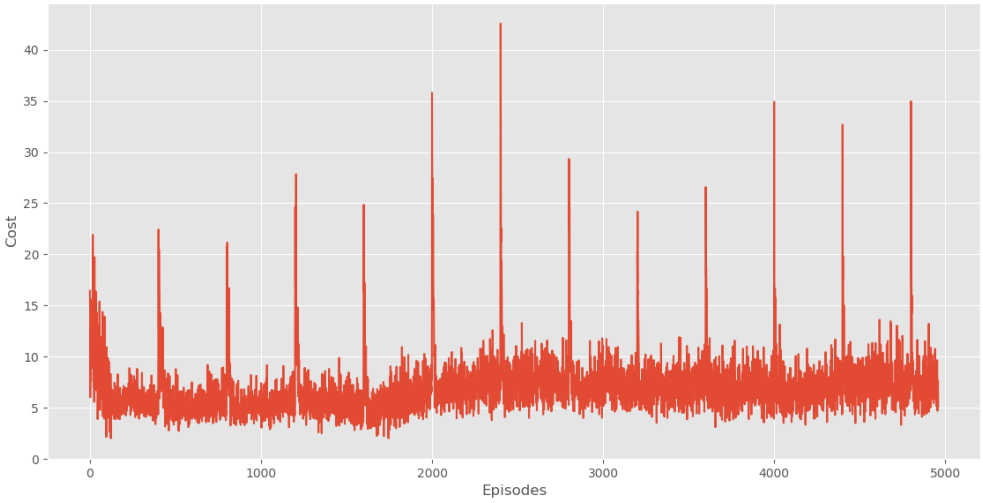


Figure 7.12: Cost Analysis of an instance in Deep Dyna-Q+ Simulation

7.4 Summary

This chapter discuss about the evaluation process which is used to evaluate the developed system. It also presents the results to analyze the performance in different scenarios. As per the evaluation results it was verified that the proposed Deep Dyna-Q+ approach incur best results than Dyna-Q+ and QL simulations for dynamic task scheduling in an uncertain environment.

Chapter 08

Conclusion & Further Work

8.1 Introduction

The evaluation process carried out on the established system was addressed in the previous chapter. This chapter gives an overall conclusion to the entire research project. Project results overview in terms of the targeted objectives, limitations encountered, and further research opportunities identified are also consideration under this chapter.

8.2 Conclusion

In the beginning of the research project, it was hypothesized that Multi Agent Reinforcement Learning and Enhanced Q Learning can be adopted to solve the problem of dynamic task scheduling in an unforeseen environment. In order to prove the hypothesis, the hybrid application was grounded up using the pre-identified technology reinforcement learning.

Firstly, the literature on the area was comparatively reviewed and the limitations in dynamic task scheduling and opportunities and research focus of reinforcement learning was identified. Considerable amount of time and effort was undertaken to understand the concept of reinforcement learning and filling the knowledge gap by reviewing research papers, reference books and knowledge obtained from e-learning courses. The design of the modules and interactions with each module began with the development phase. For the specified modules, implementation was completed. The implementation went in as several stages, firstly implementing model-free approach, secondly the model-based approach and then to the integration of model-free and model-based reinforcement learning approaches.

Finally, in Chapter 7, the evaluation process and findings were discussed. Evaluation process was undertaken using several evaluation criteria setups by the researcher in order to access the performance of the proposed approach. Based on the results of the evaluation, development enhances the target process, so the aims and objectives has been archived.

Significant objectives were also established at the initial stage of the process in order to achieve the target. The achievement of the objectives with respect to the project results have been summarized in the next sub section.

8.3 Project Results Overview

This session emphasizes the status and the sections related for the project outcomes against the project objectives.

Table 8.1: Project Results Overview

Objectives	Status	Sections Related
1. Comprehensively review the literature review on Dynamic Task Scheduling, Multi Agent Reinforcement Learning, Deep Q-Learning and Dyna Q+-Learning.	Completed	Chapter 02
2. Design and develop hybrid approach by integrating Multi Agent Reinforcement Learning and Dyna Q+ Learning and Deep Dyna Q+-Learning for dynamic task scheduling.	Completed	Chapter 04, Chapter 05
3. Implement the dynamic task scheduling using the proposed approach.	Completed	Chapter 03, Chapter 06
4. Evaluation of the developed system.	Completed	Chapter 07

Thus, all the above objectives were executed in the research as anticipated, as a concluding note overall system development and evaluation process was succeeded.

8.4 Limitations and Further Work

In the GUI based simulation, there is a system delay occurring when overloaded with multiple machine agents and many tasks. The system should be further optimized by doing further research on Reinforcement learning and optimization related applications. Furthermore, this application is mainly based on online reinforcement learning, where learning is happening in real time with the real data inputted by the user. It would be promising to further research and study on the effect of offline reinforcement learning incorporation for the solution and evaluate whether if it produces better simulation experience and results in scheduling unforeseen environments.

In way to produce the system more useful and powerful, few other items were also recognized as potential further works. One is setup the machine agents to have different performance capabilities. Other is setup the simulation on different workstations.

8.5 Summary

The conclusion, limitations and future work of the established hybrid application are outlined in this chapter. The constraints were identified, and possible solutions were also discussed. In order to make the application more useful and powerful, few addition works were defined as further works.

References

[1] Zhang D, Han X, Deng C, Review on the research and practice of deep learning and reinforcement learning in smart grids. CSEE Journal of Power and Energy Systems. 2018 Sep 24;4(3): 362-70. https://doi.org/10.17775/CSEEJPES.2018.00520.

[2] Xie J, Gao L, Peng K, Li X, Li H. Review on flexible job shop scheduling. IET Collaborative Intelligent Manufacturing. 2019 Sep 1;1(3):67–77. https://doi.org/10.1049/iet-cim.2018.0009.

[3] Luo S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Applied Soft Computing. 2020 Jun; 91:106208. https://doi.org/10.1016/j.asoc.2020.106208.

[4] Nashid Anjum MD, Wang H. Dynamic scheduling and analysis of real time systems with multiprocessors. Digital Communications and Networks. 2016 Aug;2(3): 130–8. https://doi.org/10.1016/j.dcan.2016.06.004.

[5] Hagras T, Janeček J. Static vs. Dynamic List-Scheduling Performance Comparison. Acta Polytechnica. 2003; 43(6).

[6] Kopetz H, Real-Time Scheduling. In: Real-Time Systems. The International Series in Engineering and Computer Science. vol 395. Springer: Boston, MA; 2002.

[7] Huang Z, van der Aalst WMP, Lu X, Duan H. Reinforcement learning based resource allocation in business process management. Data and Knowledge Engineering. 2011 Jan;70(1):127–45.

[8] Kumar V, Bhambri S, Shambharkar PG. Multiple Resource Management and Burst Time Prediction using Deep Reinforcement Learning. In: Eighth International Conference on Advances in Computing, Communication and Information Technology CCIT; 2019.p. 51-58.

[9] Xiao Z, Ma S, Zhang S. Learning Task Allocation for Multiple Flows in Multi-Agent Systems. In: 2009 International Conference on Communication Software and Networks, Chengdu Sichuan, China; 2009. p. 153–7. https://doi.org/10.1109/ICCSN.2009.28.

[10] Zhao X, Zong Q, Tian B, Zhang B, You M. Fast task allocation for heterogeneous unmanned aerial vehicles through reinforcement learning. Aerospace Science and Technology. 2019 Sep;92: 588–94. https://doi.org/10.1016/j.ast.2019.06.024 .

[11] Nguyen H, La H. Review of Deep Reinforcement Learning for Robot Manipulation. In: 2019 Third IEEE International Conference on Robotic Computing (IRC). Naples, Italy; 2019. p. 590-5. https://doi.org/10.1109/IRC.2019.00120.

[12] Hou J, Li H, Hu J, Zhao C, Guo Y, Li S et al. A review of the applications and hotspots of reinforcement learning. In: 2017 IEEE International Conference on Unmanned Systems (ICUS). Beijing, China; 2017. p. 506-11. https://doi.org/10.1109/ICUS.2017.8278398.

[13] Zhang L, Qi Q, Wang J, Sun H, Liao J. Multi-task Deep Reinforcement Learning for Scalable Parallel Task Scheduling. In: 2019 IEEE International Conference on Big Data (Big Data). Los Angeles, CA, USA: IEEE; 2019. p. 2992–3001. https://doi.org/10.1109/BigData47090.2019.9006027.

[14] Sutton RS, Barto AG. Reinforcement Learning: An Introduction. Second Edition. The MIT Press Cambridge, Massachusetts London, England; 2018.

[15] Tian Y-T, Yang M, Qi X-Y, Yang Y-M. Multi-robot task allocation for fire-disaster response based on reinforcement learning. In: 2009 International Conference on Machine Learning and Cybernetics. IEEE; 2009. p. 2312–2317. https://doi.org/10.1109/ICMLC.2009.5212216.
[16] Arel I, Liu C, Urbanik T, Kohls AG. Reinforcement learning-based multi-agent system for network traffic signal control. IET Intelligent Transport Systems. 2010;4(2):128. https://doi.org/10.1049/iet-its.2009.0070.
[17] Wang Y-C, Usher JM. Application of reinforcement learning for agent-based production scheduling. Engineering Applications of Artificial Intelligence. 2005 Feb;18(1):73–82. https://doi.org/10.1016/j.engappai.2004.08.018.
[18] Sun Y, Tan W. A trust-aware task allocation method using deep q-learning for uncertain mobile crowdsourcing. Human-centric Computing and Information Sciences. 2019;9(1):25. https://doi.org/10.1186/s13673-019-0187-4.
[19] Ben Noureddine D, Gharbi A, Ben Ahmed S. Multi-agent Deep Reinforcement Learning for Task Allocation in Dynamic Environment: In: Proceedings of the 12th International Conference on Software Technologies, Madrid, Spain: SCITEPRESS - Science and Technology Publications; 2017. p. 17–26. https://doi.org/10.5220/0006393400170026.
[20] Zhang K, Zhu Y, Leng S, He Y, Maharjan S, Zhang Y. Deep Learning Empowered Task Offloading for Mobile Edge Computing in Urban Informatics. IEEE Internet Things J. 2019 Oct;6(5):7635–47. https://doi.org/10.1109/JIOT.2019.2903191.
[21] Chantaravarapan S, Gunal A, Williams EJ. On Using Monte Carlo Methods for Scheduling. In: Proceedings of the 2004 Winter Simulation Conference, 2004. Washington, D.C.: IEEE; 2004. p. 789–94. https://doi.org/10.1109/WSC.2004.1371542.
[22] Zhang W, Dietterich TG. A Reinforcement Learning Approach to Job-shop Scheduling. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95). Morgan Kaufmann, Orlando, FL; 1995. p. 1114–20.
[23] Zhang W. Reinforcement Learning for Job-Shop Scheduling [Doctor of Philosophy in Computer Science]. Oregon State University; 1996.
[24] Lowe R, Wu Y, Tamar A, Harb J, Abbeel P, Mordatch I. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. arXiv:170602275 [cs] [Internet]. 2017 Jun 7; Available from: http://arxiv.org/abs/1706.02275.
[25] Wei Y, Yu FR, Song M, Han Z. User Scheduling and Resource Allocation in HetNets with Hybrid Energy Supply: An Actor-Critic Reinforcement Learning Approach. IEEE Trans Wireless Commun. 2018 Jan;17(1):680–92. https://doi.org/10.1109/TWC.2017.2769644.
[26] Liu C-L, Chang C-C, Tseng C-J. Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems. IEEE Access. 2020; 8:71752–62. https://doi.org/10.1109/ACCESS.2020.2987820.
[27] Wiering M. Multi-Agent Reinforcement Learning for Traffic Light Control. In: 17th International Conf. on Machine Learning (ICML). 2000; p. 1151–8.
[28] Sutton RS. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In: Machine Learning Proceedings 1990. Elsevier; 1990. p. 216–24. https://doi.org/10.1016/B978-1-55860-141-3.50030-4.
[29] Peng B, Li X, Gao J, Liu J, Wong K-F. Deep Dyna-Q: Integrating Planning for Task-Completion Dialogue Policy Learning. In: Proceedings of the 56th Annual Meeting of

the Association for Computational Linguistics (Volume 1: Long Papers). Melbourne, Australia: Association for Computational Linguistics; 2018. p. 2182–92. https://doi.org/10.18653/v1/P18-1203.

[30] Su S, Li X, Gao J, Liu J, Chen Y. Discriminative Deep Dyna-Q: Robust Planning for Dialogue Policy Learning. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. 2018. https://arxiv.org/abs/1808.09442.

[31] Cui J, Liu Y, Nallanathan A. Multi-Agent Reinforcement Learning Based Resource Allocation for UAV Networks. IEEE Transactions on Wireless Communications. 2019 Feb;1–1. https://doi.org/10.1109/TWC.2019.2935201.

[32] Zheng L, Yang J, Cai H, Zhang W, Wang J, Yu Y. MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence. arXiv:171200600 [cs] [Internet]. 2017 Dec 2; Available from: http://arxiv.org/abs/1712.00600.

[33] Kim D, Moon S, Hostallero D, Kang WJ, Lee T, Son K, et al. Learning to Schedule Communication in Multi-agent Reinforcement Learning. arXiv:190201554 [cs] [Internet]. 2019 Feb 5; Available from: http://arxiv.org/abs/1902.01554.

[34] Gabel T, Riedmiller M. Adaptive Reactive Job-Shop Scheduling with Reinforcement Learning Agents. International Journal of Information Technology and Intelligent Computing. 2008a;24(4):30.

[35] Wu J, Xu X, Zhang P, Liu C. A novel multi-agent reinforcement learning approach for job scheduling in Grid computing. Future Generation Computer Systems. 2011 May;27(5):430–9. https://doi.org/10.1016/j.future.2010.10.009.

[36] Wu J, Xu X. Decentralised grid scheduling approach based on multi-agent reinforcement learning and gossip mechanism. CAAI Transactions on Intelligence Technology. 2018 Mar 1;3(1):8–17. https://doi.org/10.1049/trit.2018.0001.

[37] Moradi M. A centralized reinforcement learning method for multi-agent job scheduling in Grid. In 2016 6th International Conference on Computer and Knowledge Engineering (ICCKE). Mashhad, Iran. Oct. 2016. pp. 171–176. https://doi.org/10.1109/ICCKE.2016.7802135.

[38] Peng J, Williams RJ. Efficient Learning and Planning Within the Dyna Framework. :7. https://doi.org/10.1177/105971239300100403.

[39] Morales M. Grokking Deep Reinforcement Learning. Manning Publications. 2020.

[40] Shyalika C, Silva T, and Karunananda A. Reinforcement Learning in Dynamic Task Scheduling: A Review. SN COMPUT. SCI. vol. 1. no. 6. p. 306. Nov. 2020. https://doi.org/10.1007/s42979-020-00326-5.

[41] Shyalika, C. and Silva, T., 2021. Reinforcement learning based an Integrated Approach for Uncertainty Scheduling in Adaptive Environments using MARL. In: 2021 6th International Conference on Inventive Computation Technologies (ICICT). [online] Coimbatore, India: IEEE, pp.1204-1211. Available at: <https://ieeexplore.ieee.org/abstract/document/9358727>.https://doi.org/10.1109/ICICT50816.2021.9358727 .

Appendix 01:

Sample Codes

This appendix includes some of the important code snippets developed.

1. Data Acquisition Window

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1029, 729)
        MainWindow.setAutoFillBackground(False)
        MainWindow.setStyleSheet("background-color:rgb(255, 252, 217)")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.textEdit_2_no_steps = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_2_no_steps.setGeometry(QtCore.QRect(440, 190, 41, 31))
        self.textEdit_2_no_steps.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.textEdit_2_no_steps.setObjectName("textEdit_2_no_steps")
        self.rlPushButton = QtWidgets.QPushButton(self.centralwidget)
        self.rlPushButton.setGeometry(QtCore.QRect(160, 580, 111, 41))
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(True)
        font.setWeight(75)
        self.rlPushButton.setFont(font)
        self.rlPushButton.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.rlPushButton.setObjectName("rlPushButton")
        self.dynaqPushButton = QtWidgets.QPushButton(self.centralwidget)
        self.dynaqPushButton.setGeometry(QtCore.QRect(420, 580, 111, 41))
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(True)
        font.setItalic(False)
        font.setWeight(75)
        font.setStrikeOut(False)
        self.dynaqPushButton.setFont(font)
        self.dynaqPushButton.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.dynaqPushButton.setObjectName("dynaqPushButton")
        self.label_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(120, 250, 201, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(120, 80, 161, 31))
        font = QtGui.QFont()
```

```
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(120, 190, 201, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.deepdynaqPushButton = QtWidgets.QPushButton(self.centralwidget)
        self.deepdynaqPushButton.setGeometry(QtCore.QRect(640, 580, 111, 41))
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(True)
        font.setItalic(False)
        font.setWeight(75)
        self.deepdynaqPushButton.setFont(font)
    self.deepdynaqPushButton.setStyleSheet("backgroundcolor:rgb(255,255,255)")
        self.deepdynaqPushButton.setObjectName("deepdynaqPushButton")
        self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(350, 510, 261, 31))
        font = QtGui.QFont()
        font.setPointSize(11)
        font.setBold(True)
        font.setWeight(75)
        self.lineEdit.setFont(font)
        self.lineEdit.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.lineEdit.setObjectName("lineEdit")
        self.lineEdit_2 = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_2.setGeometry(QtCore.QRect(340, 10, 271, 31))
        font = QtGui.QFont()
        font.setPointSize(11)
        font.setBold(True)
        font.setWeight(75)
        self.lineEdit_2.setFont(font)
        self.lineEdit_2.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.cancel_pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.cancel_pushButton.setGeometry(QtCore.QRect(810, 650, 91, 31))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.cancel_pushButton.setFont(font)
        self.cancel_pushButton.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.cancel_pushButton.setObjectName("cancel_pushButton")
        self.plainTextEdit_no_goals = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_no_goals.setGeometry(QtCore.QRect(420, 70, 61, 41))
        self.plainTextEdit_no_goals.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_no_goals.setObjectName("plainTextEdit_no_goals")
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(360, 250, 81, 21))
        font = QtGui.QFont()
        font.setPointSize(10)
```

```
        self.label_4.setFont(font)
        self.label_4.setObjectName("label_4")
        self.label_5 = QtWidgets.QLabel(self.centralwidget)
        self.label_5.setGeometry(QtCore.QRect(470, 250, 51, 21))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.label_5.setFont(font)
        self.label_5.setObjectName("label_5")
        self.label_6 = QtWidgets.QLabel(self.centralwidget)
        self.label_6.setGeometry(QtCore.QRect(120, 140, 161, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_6.setFont(font)
        self.label_6.setObjectName("label_6")
        self.plainTextEdit_2_no_agents =
QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_2_no_agents.setGeometry(QtCore.QRect(420, 130, 61, 41))
        self.plainTextEdit_2_no_agents.setStyleSheet("background-color:rgb(255,
255, 255)")
    self.plainTextEdit_2_no_agents.setObjectName("plainTextEdit_2_no_agents")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(630, 450, 31, 21))
        self.pushButton.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.pushButton.setObjectName("pushButton")
        self.label_7 = QtWidgets.QLabel(self.centralwidget)
        self.label_7.setGeometry(QtCore.QRect(690, 240, 291, 281))
        self.label_7.setText("")
   self.label_7.setPixmap(QtGui.QPixmap("E:/MSC/FinalProject/images/new.jpg"))
        self.label_7.setScaledContents(True)
        self.label_7.setObjectName("label_7")
        self.label_8 = QtWidgets.QLabel(self.centralwidget)
        self.label_8.setGeometry(QtCore.QRect(600, 80, 91, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_8.setFont(font)
        self.label_8.setObjectName("label_8")
        self.plainTextEdit_action1 = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_action1.setGeometry(QtCore.QRect(680, 70, 51, 41))
        self.plainTextEdit_action1.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_action1.setObjectName("plainTextEdit_action1")
        self.plainTextEdit_action2 = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_action2.setGeometry(QtCore.QRect(750, 70, 51, 41))
        self.plainTextEdit_action2.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_action2.setObjectName("plainTextEdit_action2")
        self.plainTextEdit_action3 = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_action3.setGeometry(QtCore.QRect(820, 70, 51, 41))
        self.plainTextEdit_action3.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_action3.setObjectName("plainTextEdit_action3")
        self.plainTextEdit_action4 = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_action4.setGeometry(QtCore.QRect(890, 70, 51, 41))
        self.plainTextEdit_action4.setStyleSheet("background-color:rgb(255, 255,
255)")
```

```
        self.plainTextEdit_action4.setObjectName("plainTextEdit_action4")
        self.label_9 = QtWidgets.QLabel(self.centralwidget)
        self.label_9.setGeometry(QtCore.QRect(600, 150, 91, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_9.setFont(font)
        self.label_9.setObjectName("label_9")
        self.plainTextEdit_agent2 = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_agent2.setGeometry(QtCore.QRect(750, 140, 51, 41))
        self.plainTextEdit_agent2.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_agent2.setObjectName("plainTextEdit_agent2")
        self.plainTextEdit_agent3 = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_agent3.setGeometry(QtCore.QRect(820, 140, 51, 41))
        self.plainTextEdit_agent3.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_agent3.setObjectName("plainTextEdit_agent3")
        self.plainTextEdit_action4_2 =
QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_action4_2.setGeometry(QtCore.QRect(890, 140, 51, 41))
        self.plainTextEdit_action4_2.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_action4_2.setObjectName("plainTextEdit_action4_2")
        self.plainTextEdit_agent1 = QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_agent1.setGeometry(QtCore.QRect(680, 140, 51, 41))
        self.plainTextEdit_agent1.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_agent1.setObjectName("plainTextEdit_agent1")
        self.textEdit_st1 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_st1.setGeometry(QtCore.QRect(390, 280, 41, 31))
        self.textEdit_st1.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_st1.setObjectName("textEdit_st1")
        self.textEdit_st2 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_st2.setGeometry(QtCore.QRect(390, 320, 41, 31))
        self.textEdit_st2.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_st2.setObjectName("textEdit_st2")
        self.textEdit_st3 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_st3.setGeometry(QtCore.QRect(390, 360, 41, 31))
        self.textEdit_st3.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_st3.setObjectName("textEdit_st3")
        self.textEdit_rw1 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_rw1.setGeometry(QtCore.QRect(470, 280, 41, 31))
        self.textEdit_rw1.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_rw1.setObjectName("textEdit_rw1")
        self.textEdit_rw2 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_rw2.setGeometry(QtCore.QRect(470, 320, 41, 31))
        self.textEdit_rw2.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_rw2.setObjectName("textEdit_rw2")
        self.textEdit_rw3 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_rw3.setGeometry(QtCore.QRect(470, 360, 41, 31))
        self.textEdit_rw3.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_rw3.setObjectName("textEdit_rw3")
        self.textEdit_st4 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_st4.setGeometry(QtCore.QRect(390, 400, 41, 31))
        self.textEdit_st4.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_st4.setObjectName("textEdit_st4")
        self.textEdit_rw4 = QtWidgets.QTextEdit(self.centralwidget)
```

```
        self.textEdit_rw4.setGeometry(QtCore.QRect(470, 400, 41, 31))
        self.textEdit_rw4.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_rw4.setObjectName("textEdit_rw4")
        self.textEdit_st5 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_st5.setGeometry(QtCore.QRect(390, 440, 41, 31))
        self.textEdit_st5.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_st5.setObjectName("textEdit_st5")
        self.textEdit_rw5 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_rw5.setGeometry(QtCore.QRect(470, 440, 41, 31))
        self.textEdit_rw5.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_rw5.setObjectName("textEdit_rw5")
        self.label_10 = QtWidgets.QLabel(self.centralwidget)
        self.label_10.setGeometry(QtCore.QRect(540, 250, 51, 21))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.label_10.setFont(font)
        self.label_10.setObjectName("label_10")
        self.textEdit_pr1 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_pr1.setGeometry(QtCore.QRect(540, 280, 41, 31))
        self.textEdit_pr1.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_pr1.setObjectName("textEdit_pr1")
        self.textEdit_pr2 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_pr2.setGeometry(QtCore.QRect(540, 320, 41, 31))
        self.textEdit_pr2.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_pr2.setObjectName("textEdit_pr2")
        self.textEdit_pr3 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_pr3.setGeometry(QtCore.QRect(540, 360, 41, 31))
        self.textEdit_pr3.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_pr3.setObjectName("textEdit_pr3")
        self.textEdit_pr4 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_pr4.setGeometry(QtCore.QRect(540, 400, 41, 31))
        self.textEdit_pr4.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_pr4.setObjectName("textEdit_pr4")
        self.textEdit_pr2_4 = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit_pr2_4.setGeometry(QtCore.QRect(540, 440, 41, 31))
        self.textEdit_pr2_4.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.textEdit_pr2_4.setObjectName("textEdit_pr2_4")
        self.pause_pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pause_pushButton.setGeometry(QtCore.QRect(840, 540, 91, 31))
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.pause_pushButton.setFont(font)
        self.pause_pushButton.setStyleSheet("background-color:rgb(255, 255, 255)")
        self.pause_pushButton.setObjectName("pause_pushButton")
        self.plainTextEdit_action4_3 =
QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_action4_3.setGeometry(QtCore.QRect(960, 70, 51, 41))
        self.plainTextEdit_action4_3.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_action4_3.setObjectName("plainTextEdit_action4_3")
        self.plainTextEdit_action4_4 =
QtWidgets.QPlainTextEdit(self.centralwidget)
        self.plainTextEdit_action4_4.setGeometry(QtCore.QRect(960, 140, 51, 41))
        self.plainTextEdit_action4_4.setStyleSheet("background-color:rgb(255, 255,
255)")
        self.plainTextEdit_action4_4.setObjectName("plainTextEdit_action4_4")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
```

```
        self.menubar.setGeometry(QtCore.QRect(0, 0, 1029, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.rlPushButton.setText(_translate("MainWindow", "RL-Q "))
        self.dynaqPushButton.setText(_translate("MainWindow", "DYNA-Q"))
        self.label_3.setText(_translate("MainWindow", "Rewarding Criteria"))
        self.label.setText(_translate("MainWindow", "Number of Goals"))
        self.label_2.setText(_translate("MainWindow", "Number of Steps (max)"))
        self.deepdynaqPushButton.setText(_translate("MainWindow", "Deep Dyna Q+"))
        self.lineEdit.setText(_translate("MainWindow", "**Select Scheduling
Algorithm**"))
        self.lineEdit_2.setText(_translate("MainWindow", "**Select Scheduling
Parameters**"))
        self.cancel_pushButton.setText(_translate("MainWindow", "Cancel"))
        self.label_4.setText(_translate("MainWindow", "No. Sub-tasks"))
        self.label_5.setText(_translate("MainWindow", "Reward"))
        self.label_6.setText(_translate("MainWindow", "Number of Agents"))
        self.pushButton.setText(_translate("MainWindow", "+"))
        self.label_8.setText(_translate("MainWindow", "Actions"))
        self.label_9.setText(_translate("MainWindow", "Agents"))
        self.label_10.setText(_translate("MainWindow", "Priority"))
        self.pause_pushButton.setText(_translate("MainWindow", "Pause"))
```

```
class Ui(QtWidgets.QMainWindow):
    def __init__(self):
        super(Ui, self).__init__()
        uic.loadUi('main_window.ui', self)
      self.dyna_button = self.findChild(QtWidgets.QPushButton, 'dynaqPushButton')
        self.dyna_button.clicked.connect(self.dyna_buttonPressed)
        self.rl_button = self.findChild(QtWidgets.QPushButton, 'rlPushButton')
        self.rl_button.clicked.connect(self.rl_buttonPressed)
self.cl_button = self.findChild(QtWidgets.QPushButton, 'cancel_pushButton')
       self.cl_button.clicked.connect(self.cancel_buttonPressed)
self.data_text1 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_no_goals')
        self.data_text2 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_2_no_agents')
        self.action1 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_action1')
        self.action2 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_action2')
        self.action3 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_action3')
        self.action4 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_action4')
        self.agent1 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_agent1')
        self.agent2 = self.findChild(QtWidgets.QPlainTextEdit,
```

```
'plainTextEdit_agent2')
        self.agent3 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_agent3')
        self.agent4 = self.findChild(QtWidgets.QPlainTextEdit,
'plainTextEdit_agent4')
```

2. Agents and Tasks Initialization

```
class Agent:
    def __init__(self, count, status):
        self.count = count
        self.status = status
```

```
def __init__(self, tot_actions=ACTIONS, agents=AGENTS,
init_position=current_goal_pos):
    super(EnvMaze, self).__init__()
    self.action_space = tot_actions
    self.n_actions = len(self.action_space)
    self.n_agents = agents
    self.n_agents = len(self.n_agents)
```

3. Environment Initialization with reset

```
def env_reset(self):
    for xy in range(goals_count):
        self.agent_position[xy] = 0
    self.number_position = 0
    self.gain_reward = 0
```

4. Task Backlog Creation

```
def updateEnv():
    counter = 0
    sum = 0

    # Obtained list for Episodes via Steps plotting
    steps = []
    # Adding the costs for each episodes in resulted list
    sum_costs = []

    for epi in range(4000):

        env.env_reset()
        print("episode=" + str(episode))
        with open('logfile.txt', 'a') as fp:
            fp.writelines("\n" + "-----------------------------" + "\n")
            fp.writelines(" episode= " + str(episode) + "\n")

        position = [0, 0, 0]

        # Next update the count of Steps in each Episode
        i = 0
        # Next update the cost for each episode
        cost = 0

        while True:
```

```
            # RL chooses action based on environment observation
            n_action = RL.select_next_action(str(position), episode)

            # RL takes an action and get the next observation and reward
            next_position, reward, done, comp_results = env.step(n_action)
            # RL learns from this transition and calculating the cost
            cost += RL.learn(str(position), n_action, reward, str(next_position),
done)
            RL.learn(str(position), n_action, reward, str(next_position), done)
            env_model.store_transition(str(position), n_action, reward,
next_position)

            # for Random_agent & Dyna-Q agent-range(10)
            # for RL_agent use range(1)
            for n in range(10):  # learn 10 more times using the EnvModel in
model_RL.py
                ms_1, ma_1 = env_model.sample_s_a()  # ms here is a string
                mr_1, ms_11 = env_model.get_r_s_(ms_1, ma_1)
                 #state transition
                 #ms-current state, ma-current action, mr-current reward gained
                 #ms_ - next possible state
                RL.learn(ms_1, ma_1, mr_1, str(ms_11), done)

            position = next_position.copy()

            # Calculating number of Steps in the current Episode
            i += 1

            if done:
                sum = sum + reward
                 #dyna simulation-visiting real env.once in 50 episodes
                if epi % 50 == 0:
                    time.sleep(10)
                    data_out.append(sum / 50)
                    sum = 0
                    indexes.append(episode)
                    counter = counter + 1
                    env_model.get_env()
                        env_model.get_agent_env()
                steps += [i]
                sum_costs += [cost]
                break

            env_model.showMaze(ms_1, ma_1)
            state1 = eval(ms_1[1:-4])
            state2 = eval(ms_1[4:-1])
            state_a.append(state1)
            state_b.append(state2)
```

5. RL Learning

I. Deep Dyna-Q+ Learning Agent Learning

```
class DeepQNetwork:
    def __init__(
            self,
```

```
            tot_actions, tot_features, n_agents, learning_rate=0.1,
            reward_decay=0.9, replace_target_iterations =500,
             e_greedy=0.9, batch_size=64,
            memory_size=500, output_graph=True,
            e_greedy_increment=None,
    ):
        self.tot_actions = tot_actions
        self.tot_features = tot_features
        self.gamma = reward_decay # discount factor
         self.lr = learning_rate
        self.max_epsilon = e_greedy
        self.replace_target_iterations = replace_target_iterations
         self.epsilon = 0 if e_greedy_increment is not None else self.max_epsilon
        self.memory_size = memory_size    # replay buffer size
        self._epsilon_increment = e_greedy_increment
        self.batch_size = batch_size  # minibatch size
        self.n_agents = n_agents
```

```
def build_dqn(self):
    # ------------------ building evaluate_net ------------------
    self.s = tf.compat.v1.placeholder(tf.float32, [None, self.tot_features],
name='s')  # input for the dqn-states
    self.q_target = tf.compat.v1.placeholder(tf.float32, [None, self.tot_actions],
                                name='Q_target')  # for calculating loss
    with tf.compat.v1.variable_scope('eval_net'):
        # c_names(collections_names) are the collections to store variables
        c_names, n_l1, w_initializer, b_initializer = \
            ['eval_net_params', tf.compat.v1.GraphKeys.GLOBAL_VARIABLES], 10, \
            tf.random_normal_initializer(0., 0.3), tf.constant_initializer(0.1)  #
config of layers

        # this is the first layer. Here collections are used later when assign
them to target net
        with tf.compat.v1.variable_scope('l1'):
            w1 = tf.compat.v1.get_variable('w1', [self.tot_features, n_l1],
initializer=w_initializer, collections=c_names)
            b1 = tf.compat.v1.get_variable('b1', [1, n_l1],
initializer=b_initializer, collections=c_names)
            # tf.matmul - multiply 2 matrixes
            l1 = tf.nn.relu(tf.matmul(self.s, w1) + b1)

        # this is the second layer. Here collections are used later when assign
them to target net
        # take output of first layer - l1 as input
        with tf.compat.v1.variable_scope('l2'):
            w2 = tf.compat.v1.get_variable('w2', [n_l1, self.tot_actions],
initializer=w_initializer, collections=c_names)
            b2 = tf.compat.v1.get_variable('b2', [1, self.tot_actions],
initializer=b_initializer, collections=c_names)
            self.q_eval = tf.matmul(l1, w2) + b2

    # calculating the loss function
    with tf.compat.v1.variable_scope('loss'):
        # loss=squared difference between q value and q target
        self.loss = tf.reduce_mean(tf.math.squared_difference(self.q_target,
self.q_eval))
        print("loss", self.loss)
    # define training operation. It is a form of gradient decent. we use
```

```
RMSPropOptimizer here.
    with tf.compat.v1.variable_scope('train'):
        self.train_operation =
tf.compat.v1.train.RMSPropOptimizer(self.lr).minimize(
            self.loss)  # here we need to minimize the loss function
        print("train_operation", self.train_operation)

    # ------------------ building target_net ------------------
    self.s_ = tf.compat.v1.placeholder(tf.float32, [None, self.tot_features],
name='s_')  # input
    with tf.compat.v1.variable_scope('target_net'):
        # c_names(collections_names) are the collections to store variables
        # GraphKeys.GLOBAL_VARIABLES-tell tf to keep track of all the variables
        c_names = ['target_net_params', tf.compat.v1.GraphKeys.GLOBAL_VARIABLES]

        # this is the first layer. Here collections are used later when assign to
target net
        with tf.compat.v1.variable_scope('l1'):
            w1 = tf.compat.v1.get_variable('w1', [self.tot_features, n_l1],
initializer=w_initializer, collections=c_names)
            b1 = tf.compat.v1.get_variable('b1', [1, n_l1],
initializer=b_initializer, collections=c_names)
            l1 = tf.nn.relu(tf.matmul(self.s_, w1) + b1)
            print("c_names1", c_names)

        # this is the second layer. Here collections are used later when assign
them to target net
        with tf.compat.v1.variable_scope('l2'):
            w2 = tf.compat.v1.get_variable('w2', [n_l1, self.tot_actions],
initializer=w_initializer, collections=c_names)
            b2 = tf.compat.v1.get_variable('b2', [1, self.tot_actions],
initializer=b_initializer, collections=c_names)
            self.q_next = tf.matmul(l1, w2) + b2
            print("c_names2", c_names)
```

```
# defining the learing step size in each epoch/episode
self.step_size = 0

# initializing zero memory [state, action, reward, next_state]
self.memory = np.zeros((self.memory_size, tot_features * 2 + 2))

def agents_learn(self):
    #timeweight to k initialization
    self.timeWeight = 1e-1
    # checking to replace target parameters
    if self.step_size % self.replace_target_iter == 0:
        self.sess.run(self.replace_target_op)

    # sampling batch memory from all memory
    if self.memory_counter > self.memory_size:
        sample_index = np.random.choice(self.memory_size, size=self.batch_size)
    else:
        sample_index = np.random.choice(self.memory_counter, size=self.batch_size)
    batch_memory = self.memory[sample_index, :]

    q_next, q_eval = self.sess.run(
        [self.q_next, self.q_eval],
```

```
        feed_dict={
            self.s_: batch_memory[:, -self.tot_features:],  # fixed parameters
            self.s: batch_memory[:, :self.tot_features],  # newest parameters
        })

    # change q_target w.r.t q_evaluation action
    q_target = q_eval.copy()  # copy the network, bcs we want the loss of all the
non-optimal actions to be zero

    batch_index = np.arange(self.batch_size, dtype=np.int32)
    eval_act_index = batch_memory[:, self.tot_features].astype(int)
    reward = batch_memory[:, self.tot_features + 1]
    q_target[batch_index, eval_act_index] = reward + self.gamma * np.max(q_next,
axis=1)

#Dyna Q+ implementation goes here. Addition of extra reward for unseen environment
states.
    q_target += self.timeWeight * np.sqrt(time.time() - (time.time() - 100))

    # train evaluation network
    _, self.cost = self.sess.run([self.train_operation, self.loss],
     feed_dict={self.s: batch_memory[:, :self.tot_features],self.q_target:
q_target})
    self.cost_his.append(self.cost)

    # increasing epsilon
    self.epsilon = self.epsilon + self.epsilon_increment if self.epsilon <
self.max_epsilon else self.max_epsilon
    self.step_size += 1
```

II. Dyna-Q+ Agent Learning

```
def select_next_action(self, observation, episode):
    self.if_state_exist(observation)
    # act_s selection based on observation
    if self.agent == "RANDOM_LEARNING_AGENT":
        act_s = np.random.choice(self.n_actions)
        return act_s
    if np.random.uniform() < self.epsilon:
        # Exploitation starts
        # choose best action - taking the biggest Q value for this state
        state_action = self.q_table.ix[observation, :]
        state_action =
state_action.reindex(np.random.permutation(state_action.index))
        print("state_action", state_action, "\n")
        max_value = 0
        for act in list(self.q_table.columns.values):
            if self.q_table.ix[observation, act_s] >= max_value:
                max_action = act
                max_value = self.q_table.ix[observation, act_s]
        act_s = max_action
    else:
        # Exploration starts
        # agents choose random action
        act_s = np.random.choice(self.n_actions)

    # reduce epsilon because we need less and less exploration -used e-epsilon
greedy action selection mechanism
```

```
    self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon) *
np.exp(-self.decay_rate * episode)
    episode += 1
    # logging results
    print("Episode: ", episode, " ", "Epsilon: ", self.epsilon)
```

6. Priority Based Task Allocation

```
import heapq

for a in subtasks_and_rewards :
    heapq.heappush(q, (
        subtasks_and_rewards [x][2], subtasks_and_rewards [x]))
    x = x + 1

subtasks_and_rewards= []
while q:
    next_item = heapq.heappop(q)
    subtasks_and_rewards+= (next_item[1],)
```

```
def step_priority(self, action_id):
    for agent in range(self.agents.size):
        #keep track of task completions
        task_compl= 0
        self.position[action_id] = self.position[action_id] + 1
        self.position_no = self.position_no + 1
        done = False
        if self.position_no >= No_of_steps:
            done = True
            task_compl = 0
            for g in range(Number_of_goals):
                if self.is_completed_priority(g) and self.agent_state_empty():
                    self.reward_gain = self.reward_gain + subtasks_and_rewards
[g][1]
                    task_compl = task_compl + 1
                    self.step(action_id)
        action_id
```

7. Agent Message Passing

```
def store_transitions(self,state,action,reward,next_state, agent_state):
    if not hasattr(self, 'memory_counter'):
        self.memory_counter = 0  # keeps tract of the no.of memories we store

    transition = np.hstack(
        (state,[action,reward],next_state))  # numpy.hstack() function is used to
stack the sequence of input arrays horizontally

    # replacing old memory with new memory
    # the agent has some fixed memory size and we want to fill up to that memory
and when we exceed the memory, we want to go to beginning and start up again
overridden it
    index = self.memory_counter % self.memory_size

    self.memory[index, :] = transition
```

```
    self.memory_counter += 1

    print("transition", transition)
    print("Current database", (state, [action, reward],next_state))
    print(utils.colorize(tabulate((state,next_state), headers='keys',
tablefmt='psql'), "blue", highlight=True))
    transition2 = [state,next_state]
    # agent_state = agent_state[1]
    agent = [state,next_state], "Agent: ", (find_agent(state,t),
np.random.get_free_agent(0, self.n_agents))
    print(agent)
    with open('agent.txt', 'a') as fp:
        fp.write(str(agent) + '\n')
```

```
def find_agent(self, arr, time):
    n = len(arr)

    # Keeping track of free machine agents
    agent = [False] * time

    # Storing agent (Sequence of tasks/jobs)
    job = ['-1'] * time

    # Iterating through all inserted tasks/jobs
    for i in range(len(arr)):

        # to find a free machine agent for this task/job
        # here we start from the # last possible agent
        for j in range(n_agents):

            # Free agent found
            if agent[j] is False:
                agent[j] = True
                job[j] = arr[0]
                break
```

8. Printing Maze View

```
def showMaze(self, state, action):
    print('-------------------------------------')
    out = '| '
    for j in self.database:
        if j == 0:
            token = '0'
        if j == 1:
            token = '1'
        if j == 2:
            token = '2'
        out += token + ' | '
        if j == 3:
            token = '3'
        out += token + ' | '
        # iterate over episodes for all agents
        print(out)

    print('-------------------------------------')
    return out
```

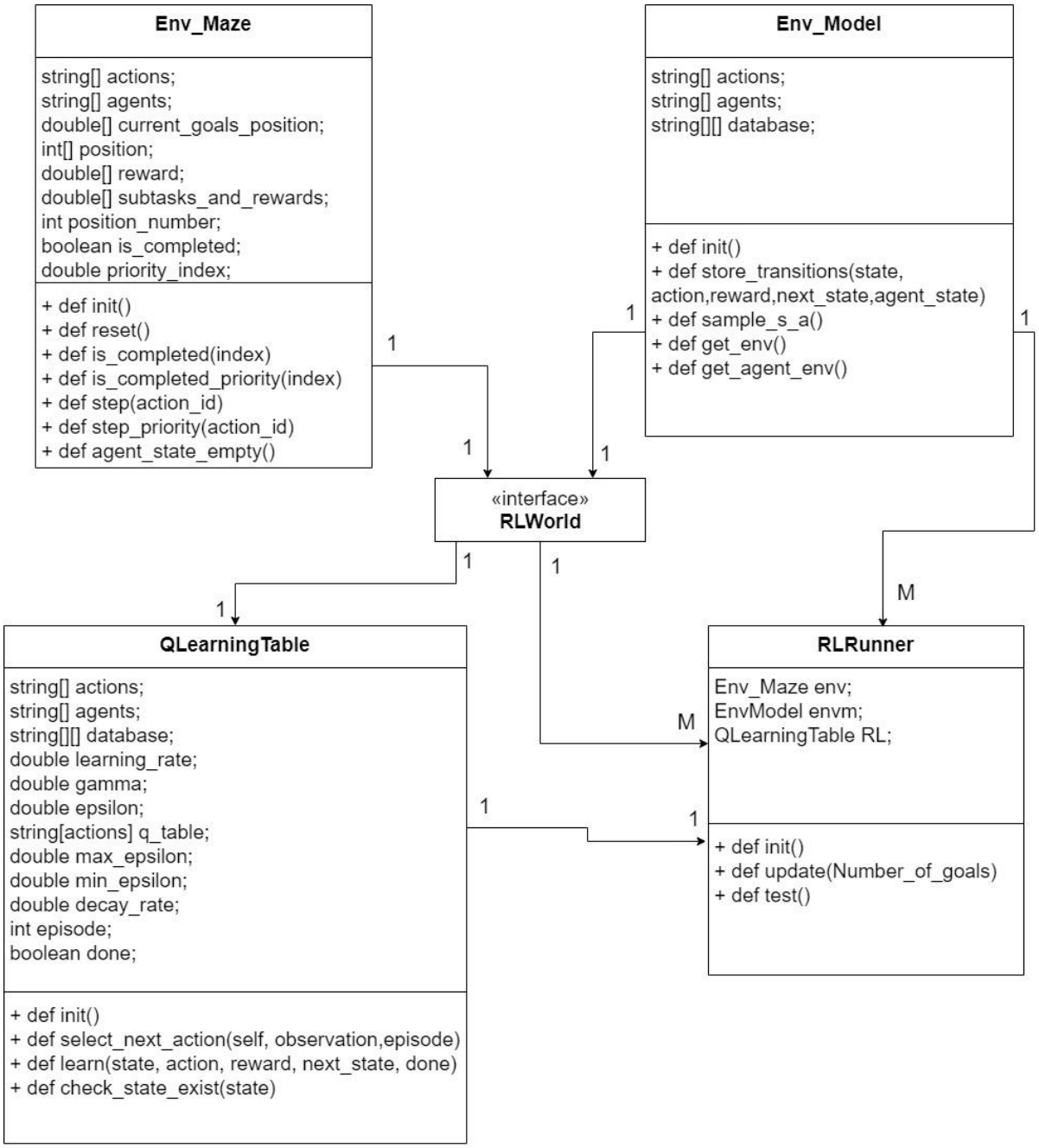
9. Generating the Schedule

```
import plotly.express as sc_plx
import pandas as pd
import os

def draw_schedule(self):
    df = pd.read_csv('allocated_tasks.csv')

    sc_fig = sc_plx.timeline(df, x_start="Start", x_end="Finish", y="Resource",
color="Task")

    sc_fig.update_layout(xaxis=dict(
        title='Timestamp',
        tickformat='%H:%M:%S',
    ))
    if not os.path.exists("images"):
        os.mkdir("images")
    sc_fig.write_image("images/schedule.png")
    sc_fig.show()
```

Appendix 02:

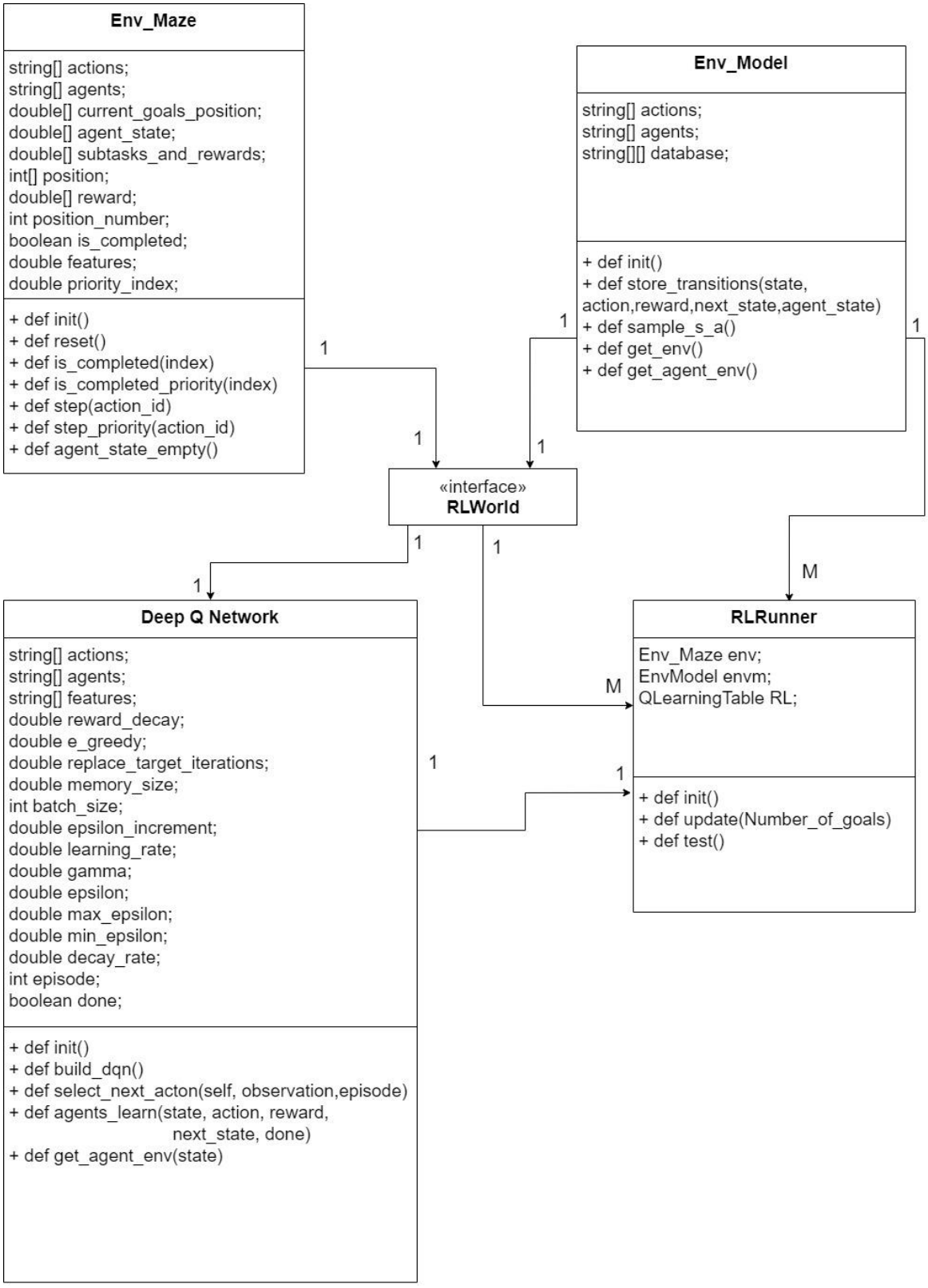
Class Diagram

I. Class Diagram for Dyna Q+ Learning



Appendix 02.A: Class Diagram for Dyna Q+ Learning

II. Class Diagram for Deep Dyna Q+ Learning



Appendix 02.B: Class Diagram for Deep Dyna Q+ Learning

Appendix 03:

Publications

I. Review Paper Published in SN Computer Science-Springer Nature
Link: https://link.springer.com/article/10.1007/s42979-020-00326-5

SN Computer Science (2020) 1:306
https://doi.org/10.1007/s42979-020-00326-5

REVIEW ARTICLE

Reinforcement Learning in Dynamic Task Scheduling: A Review

Chathurangi Shyalika[1] · Thushari Silva[1] · Asoka Karunananda[1]

Received: 12 July 2020 / Accepted: 8 September 2020

Abstract
Scheduling is assigning shared resources over time to efficiently complete the tasks over a given period of time. The term is applied separately for tasks and resources correspondingly in task scheduling and resource allocation. Scheduling is a popular topic in operational management and computer science. Effective schedules ensure system efficiency, effective decision making, minimize resource wastage and cost, and enhance overall productivity. It is generally a tedious task to choose the most accurate resources in performing work items and schedules in both computing and business process execution. Especially in real-world dynamic systems where multiple agents involve in scheduling various dynamic tasks is a challenging issue. Reinforcement Learning is an emergent technology which has been able to solve the problem of the optimal task and resource scheduling dynamically. This review paper is about a research study that focused on Reinforcement Learning techniques that have been used for dynamic task scheduling. The paper addresses the results of the study by means of the state-of-the-art on Reinforcement learning techniques used in dynamic task scheduling and a comparative review of those techniques.

Keywords Task scheduling · Reinforcement learning · Multi-agent · Dynamic · Environment uncertainty

Introduction

Task allocation ensures that the correct resources have been allocated effectively to perform work items/tasks of a particular consequence at the right time. It guarantees the balance between the demand for process execution facilities against the availability of these resources. The allocation of tasks or resources, also known as scheduling, is applicable to a number of applications, such as Industrial Workforce Management, Grid Computing, Public Transport and Network Routing. Dynamic task scheduling has gained significant attention in these fields as it is crucial for effective resource sharing.

Scheduling in a computer system is done by the component named Scheduler, which mainly concerns throughput, latency and response time. Throughput refers to how fast it could finish a certain number of tasks from beginning to end per unit of time. In contrast, latency is the turnaround time or the time it takes to complete the job from the time of request or submission until the finish, that includes the waiting time before it could be served. Response time is the time it has taken for the process or request to be served, in short, the waiting time.

Job-Shop Scheduling Problem (JSSP) [1–4] is a common problem of optimization in the field of computer science, where resources are distributed at different times by the most suitable jobs. A Job Shop can be described as a work location comprised of several individual general purpose work stations that are responsible in performing several varied tasks/jobs. JSSP can be defined as, finding the best scheduling criteria for the given 'n' jobs J1, J2,..., Jn of varying processing times, that need to be scheduled on 'm' machines with adaptable processing powers. The best solution of JSSP must be able to minimize the makespan, that is the total length of the schedule (that is when all the jobs have finished processing) [3, 4]. On each of the specific machine, there can be sub-tasks of the major tasks that are processed under a predefined specific order/sequence.

Scheduling solutions can be divided into major two categories as Static and Dynamic Scheduling [5, 6]. In

✉ Chathurangi Shyalika
chathurangijks@gmail.com

Thushari Silva
thusharip@uom.lk

Asoka Karunananda
askarunananda@gmail.com

[1] Department of Computational Mathematics, Faculty of Information Technology, University of Moratuwa, Katubedda, Sri Lanka

Published online: 24 September 2020

Appendix 03.A: First Page of the Review Paper

II. Research Paper Presented at ICICT 2021 Conference
Link: http://icicts.com/2021/

An Integrated Approach for Uncertainty Scheduling in Adaptive Environments using MARL

Chathurangi Shyalika
Department of Computational Mathematics, Faculty of Information Technology
University of Moratuwa
Katubedda, Sri Lanka
chathurangiks@gmail.com

Thushari Silva
Department of Computational Mathematics, Faculty of Information Technology
University of Moratuwa
Katubedda, Sri Lanka
thusharip@uom.lk

***Abstract*— Scheduling is a universal theme being conferred in technological areas like computing and strategic areas like operational management. The core idea behind scheduling is the distribution of shared resources across time for competitive tasks. Optimization, efficiency, productivity and performance are the major metrics evaluated in scheduling. Effective scheduling under uncertainty is tricky and unpredictable and its an interesting area to study. Environmental uncertainty is a challenging extent that effect scheduling based decision making in work environments where environment dynamics subject to numerous fluctuations frequently. Reinforcement Learning is an emerging field extensively research on environmental modelling under uncertainty. Optimization in dynamic scheduling can be effectively handled using Reinforcement learning. This paper presents a research that accompanies Multi-agent reinforcement learning (MARL) and extends Q-learning, which aims to provide a solution for the dynamic task scheduling in an uncertain environment. The proposed solution approaches as a model-based Deep Dyna-Q+ algorithm-based hybrid solution to solve the scheduling problem in an unexpected complex environment. The Dyna-Q+ based scheduler includes priority-indexing based task scheduler which allocates jobs to individual cooperative agents sequentially in adaptive mode. The proposed model-based solution is comparatively evaluated against benchmark methods; Q-Learning and Dyna-Q+ learning algorithms for dynamic task scheduling. The initial evaluation results verify that the approach would be an effective solution for scheduling optimization in a dynamic and complex environment.**

Keywords—Reinforcement learning, Dynamic task scheduling, Model-based approach, Multi-agent RL, Uncertainty scheduling

I. Introduction

The process of assigning the most appropriate resources to the workstations or agents at the right time such that the operation of a system is seamlessly continued without any interruption, can be simply called as Task Allocation. The aim of effective scheduling is to optimize the output with defined inputs while balancing demand against available resources or perhaps to reduce the amount of input that is consumed to generate the necessary output. The allocation of tasks or resources, better known as scheduling, is a widespread norm in areas like industrial workforce management, grid computing, public transport and network routing. Job-Shop Scheduling Problem (JSSP), Open-shop scheduling and Flow-shop scheduling [1-4] are widely debated optimization related scheduling problems in computer science, where scheduling applications are ranging from CPU scheduling, Network scheduling, I/O scheduling to high end computer based production simulations. Computer Science based optimization has occupied with various scheduling algorithms that elegantly decide on the order of optimal resource allocation in multi-processing and multi-tasking structures. First Come First Served (FCFS), Shortest-Job-First (SJF), Priority Scheduling, Round Robin (RR), Multilevel Queue Scheduling and Multilevel Feedback Queue Scheduling are among the mostly discussed. Computer based scheduling solutions can be long term, short term or either medium term. The decision making in scheduling can be performed statically or dynamically, where dynamic task scheduling is extra promising to set up in an uncertain work environment where behavior, adaptation, knowledge, representation varies and unpredicted [5-6].

Genetic algorithms, Artificial neural networks, Fuzzy logic being the most prominent AI techniques researched to provide stable solutions for dynamic task scheduling [7-11]. Reinforcement Learning(RL) is a hot topic today and is a convincing paradigm been researched from games to high end simulations to solve variety of real world problems. It is the process of trial-and-error training for an agent/s to reach a goal in an environment while incentivizing it with a combination of rewards and penalties. RL is well suited in exploring emergent behavior due to its fascinating ability to capture environmental uncertainty while self-learning. Experience replay or ability to model different priority tasks while storing the perceptions of agents at each point of time is one of the significant advantage of Reinforcement learning [12].

This paper reports the research on a Hybrid Approach for Dynamic Task Scheduling in Uncertain Environments using the techniques; Multi-Agent Reinforcement Learning (MARL) and Enhanced Q-Learning. The solution comes with an approach for priority-based dynamic task scheduling. Enhanced Q-Learning includes developed algorithm approaches; Q-Learning, Dyna-Q+ Learning and Deep Dyna-Q+ Learning which is proposed as an effective methodology for scheduling problem. The novelty of the solutions resides on implementation of Multi agent model-based algorithmic approach utilizing Deep Dyna-Q+ Learning for dynamic task scheduling in an uncertain environment. The end solution would comparatively evaluate the product using evaluation metrics in each of the three Q-Learning variations developed. Section II of this paper addresses early research attempts at complex task planning focused on improving learning, research results and established issues. Section III offers the Methodology & Experimental Design of the proposed framework. Section IV is followed by the Results and Evaluation of the developed solution. Finally, Section V of the paper concludes by indicating further work and the research challenges identified while the development of the solution.

Appendix 03.B: First Page of the Research Paper