Ministry of Higher Education
and scientific research
University of Tunis Al Manar
higher institute of computer science

END-OF-STUDIES INTERNSHIP REPORT

Presented in order to obtain the

National Diploma in Computer Science

Specialty : Computer Science

By:

Joumene Ben Said

Implementation of a reinforcement learning model for real-time tasks placement

Professional supervisor: **Ms.Bakhta Houari** Assistant
Academic supervisor: **Ms.Bakhta Houari** Assistant

Work proposed and elaborated within ISI

Academic Year 2021 - 2022

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH
UNIVERSITY OF TUNIS EL MANAR
HIGHER INSTITUTE OF COMPUTER SCIENCE

END-OF-STUDIES INTERNSHIP REPORT

Presented in order to obtain the

National Diploma in Computer Science

Specialty : Computer Science

By:

Joumene Ben Said

Implementation of a reinforcement learning model for real-time tasks placement

Professional supervisor: **Ms.Bakhta Houari** Assistant

Academic supervisor: **Ms.Bakhta Houari** Assistant

Work proposed and elaborated within ISI

Academic Year 2021 - 2022

I authorize the students to submit their internship report for evaluation.

Professional supervisor, **Ms Bakhta Haouari**

Signature

I authorize the students to submit their internship report for evaluation.

Academic supervisor, **Ms Bakhta Haouari**

Signature

Dedications

I dedicate this work to :

My beloved mother, for her love, care and dedication. She has always believed in me, supported and guided me.

My sister, for all her advice, affection and encouragements.

My brother, my family, and all my friends who always gave me a hand whenever I needed it and to all my teachers during my educational journey.

Anyone who has engraved my life by a word that has oriented me to the good way.

Joumene Ben Said

ACKNOWLEDGEMENTS

First of all, I would like to thank "GOD".

Before starting any development, it seems appropriate to begin with sincere thanks to the people who have been kind enough to participate in this effect from near or far.
I would like to express my immense gratitude to my Professor Ms.Haouari Bakhta. It was a real privilege and honor for me to share her exceptional knowledge but also of her extraordinary human qualities would like to thank her for her supportive and helpful advice throughout the duration of this project.
I would also like to express our deepest respects and gratitude to the management of The Higher Institute of Computer Science of Ariana .I address a very particular thanks to Ms.Imen Ben hafaiedh.

Finally, I wish to thank my friends for their support and encouragement throughout our study.

Contents

General Introduction **1**

1 General Context **2**

1.1 Introduction . . . 3

1.2 Host Organization . . . 3

1.2.1 The Higher Institute of Computer Science . . . 3

1.2.2 University of Tunis El Manar . . . 4

1.3 Basic Concepts . . . 4

1.3.1 Real-Time Systems . . . 4

1.3.2 Architecture Types . . . 9

1.3.3 Refactoring . . . 10

1.3.4 NP-Complete Problems . . . 10

1.4 Problematic . . . 11

1.4.1 Models formalization . . . 12

1.4.2 Real-time feasibility . . . 13

1.5 Conclusion . . . 14

2 Reinforcement learning state of the art **15**

2.1 Introduction . . . 16

2.2 Artificial Intelligence . . . 16

2.2.1 Definition Of Artificial intelligence . . . 16

2.2.2 History Of Artificial intelligence . . . 16

2.2.3 Some Sub-domains Of Artificial intelligence . . . 17

2.3 Machine Learning . . . 18

2.3.1 Supervised Learning . . . 18

2.3.2 Unsupervised Learning . . . 20

2.4 Reinforcement Learning . . . 21

2.4.1 Markov Decision Process . . . 22

2.4.2 Episodic and Continuous Tasks: . . . 23

2.4.3 How to solve a reinforcement learning problem? . . . 23
2.4.4 Reinforcement learning Algorithms: . . . 28
2.4.5 Model-free algorithms . . . 31
2.4.6 Q-learning . . . 32
2.5 Conclusion . . . 39

3 Reinforcement Learning Model for The Task Placement Problem 40
3.1 Introduction . . . 41
3.2 Q-learning Model . . . 41
3.3 Algorithms . . . 43
3.3.1 Case study . . . 45
3.3.2 Analysing The Q-table . . . 49
3.3.3 Experimentation and evaluation . . . 50
3.4 Conclusion . . . 54

General Conclusion 55

Bibliography 56

List of Figures

1.1 Utility of the Results Produced by a Soft Real Time Task as a Function of Time . . 5
1.2 Hard Real-Time System 6
1.3 Utility of Result of a Firm Real-Time Task with Time 6
1.4 Real Time Task Model 7
1.5 Apariodic Task 8
1.6 Sporadic Task 8
1.7 All Possible Placements 11

2.1 Training data set 19
2.2 supervised learning example 19
2.3 Unsupervised learning classification example 20
2.4 RL Diagram 22
2.5 episodic tasks 23
2.6 example 1 25
2.7 Policy Types 25
2.8 Solving a Reinforcement learning problem approach 28
2.9 Model-based approach 29
2.10 Model-free approach 30
2.11 Reinforcement learning algorithms 31
2.12 model free algorithms. 32
2.13 Mario Game. 33
2.14 Game Rewards. 33
2.15 Exploration and Exploitation with epsilon-greedy 35

3.1 Initial State Q-values. 49
3.2 How The Agent Iterates the Q table. 50
3.3 Number of States 50
3.4 Number of State in a Case with 2 Processors. 51
3.5 Number of State in a Case With 3 Tasks. 51
3.6 The Execution Time Taken to Generate The States of a (m,n). 52

3.7 The Execution Time Taken to Generate The States VS The States Number In a Case With 3 processors. 52
3.8 Execution Time for initial placement and Refactoring in a Case with 5 processors. . 53
3.9 Execution Time for initial placement and Refactoring in a Case with 3 processors. . 53
3.10 Memory Space Used To Generate the Sates. 54

List of Tables

1.1 Tasks Parameters. 11
1.2 The Processors Utilization in Every Placement. 12

3.1 Task model tabular description of the considered case study 46
3.2 Deployment Model. 47
3.3 Tasks new features. 48
3.4 Deployment Model. 48

Liste des abréviations

— **AI** = **A**rtificial **I**ntelligence

— **ML** = **M**achine **L**earning

— **RL** = **R**einforcement **L**earning

— **RT** = **R**eal **T**ime

General Introduction

Real-Time systems span several domains of computer science. Networked multimedia systems, defense and space systems,embedded automotive electronics... etc. The correctness of a real-time system depends not only on the accuracy of the calculations but also on the time at which the results are produced (temporal constraints). A real-time system is not a "fast" system but a system that satisfies temporal constraints. These systems can be implemented as uniprocessors, multiprocessors, or distributed systems. It also can be hard or soft or firm systems.

Our discussion in this research concentrates on hard real-time application's tasks placement on a distributed platform.

In hard real-time systems, if any of the system's tasks misses a deadline it will cease to function and it's considered a system failure. Tasks scheduling is very complicated that's why we are going to focus first on tasks placement.

There are many possible placements of the tasks in the different processors of a distributed architecture. The number of possible placements grows exponentially with the number of tasks and processors to find an optimal placement, making this placement problem an NP-hard problem.

In order to solve NP-hard problems, there are many used approaches like heuristic methods and approximation algorithms. We will use reinforcement learning to solve this NP-hard combinatorial optimization problem.

This report encloses three chapters. In the first one we presented the host organization, certain terminologies and specified the problematic and the specification of our project. In the second chapter, we presented the field of artificial intelligence and went through reinforcement learning in detail. In the third and last chapter of this report we describe our solution model built on a Q-learning algorithm. We also tested the model using a specific use case and reviewed the given results. This report ends by a conclusion that summarizes our work and proposes other possible solution to our problem.

Chapter 1

General Context

Plan

1 **Introduction** 3

2 **Host Organization** 3

3 **Basic Concepts** 4

4 **Problematic** 11

5 **Conclusion** 14

1.1 Introduction

In this chapter, We will explain the overall context of the project. We'll begin by introducing the host organization. Then we will define certain concepts that are necessary for comprehending the problematic. Finally, we will present the problem in detail.

1.2 Host Organization

This Research subject was given to me by The Higher Institute of Computer Science as my end-of-studies projects to get National Diploma of Applied Licence in Science and Technology. In this section I'm going to present The Higher Institute of Computer Science.

1.2.1 The Higher Institute of Computer Science

The Higher Institute of Computer Science of the University of Tunis El Manar was created by decree N° 1912 of August 14, 2001. It offers the following training cycles:

— Training cycle in Computer Science License of a duration of three years;

— Training cycle of Engineers in Computer Science of a duration of three years;

— Training cycle for a Professional Master's Degree in Computer Science "Security of Communicating and Embedded Computer Systems" (SSICE);

— Training cycle in Professional Master in Computer Science "Free Software" (MP2L) in collaboration with the Virtual University of Tunis;

— Training cycle in Professional Master in Computer Science "Free Software" (MP2L) in collaboration with the Virtual University of Tunis;

— Training cycle for a Master's degree in Computer Science Research with two courses: SIIVA and GL.

The admission to the Bachelor cycle is done by the baccalaureate's orientation .

The admission to the engineering cycle is done at a level Bac + 3 in Computer Science (or equivalent) and following the specific national competition.

Admission to the Master's program is open to holders of a Bachelor's degree in Computer Science or equivalent.[1]

1.2.2 University of Tunis El Manar

The University of Tunis El Manar is a public establishment with administrative character created in 1987 under the name of "University of sciences, techniques and medicine of Tunis" by the article 97 of the law n°87-83 of December 31, 1987. It got its current name according to the decree n° 2000-2826 of November 27, 2000, concerning the change of name of universities.

Regrouping the oldest and most prestigious institutions of higher education and research in Tunisia. The UTM has a training offer of quality, open to the needs of society and multidisciplinary (basic sciences and engineering techniques, humanities, political, legal and economic sciences, medical and paramedical sciences).

With the largest number of research structures on a national scale, the UTM is both strong and radiant with quality research that covers the most varied fields (health, water, environment, energy, etc.) and is in line with the research strategy on a national and international scale [2].

It is ranked by U.S. News World Report as 11th in the 2016 regional ranking of Arab universities. According to the 2019 Shanghai ranking, the university ranks first in the Maghreb, tenth in the Arab world and eleventh in Africa. It is also included in the list of the 500 best universities in the specialties of clinical medicine and biotechnology [3].

1.3 Basic Concepts

In this section we are going to introduce some concepts that are essential for the understanding of the rest of the work.

1.3.1 Real-Time Systems

Real-time computer systems differ from other computer systems in that they take into account time constraints, the respect of which is as important as the accuracy of the result. In other words, the system must not simply deliver accurate results, it must deliver them within a given time frame [4].

Real-time computing systems are present today in many sectors of activity: in the production industry, for example, through process control systems (factories, nuclear power plants), in trading rooms through the processing of stock market data in "real time", in aeronautics through on-board piloting systems (aircraft, satellites), or in the new economy sector through the ever-increasing need for information processing and routing (video, data, remote piloting, virtual reality, etc.).

The development of real-time systems therefore requires that each of the elements of the system is itself real-time, allows to take into account temporal constraints. An operating system designed in this way is called a real-time operating system.
To be capable of real-time computing, the correctness of the computations must depend on these to requirements:

1 Functional Correction: The logical correctness of the computation.

2 Time Correction: The time at which the result is produced the deadline.

1.3.1.1 Real-Time System Categories

Real-time system are designated as either a soft real-time system , hard real-time system or firm real-time system based on timing constraints.

1 **Soft Real-Time System**

Missing the deadline in soft real-time systems does not result in system failure but in a decrease in the quality of service. When a large number of tasks miss their deadlines, the system's performance is considered to have degraded.

For example, while browsing the web when we click on a URL the associated web page is normally fetched and displayed within a few seconds, but sometimes it takes a while, when this happens we do not consider the system to have failed instead we consider that the quality of service is bad. Other examples are audio and video delivery software for entertainment.

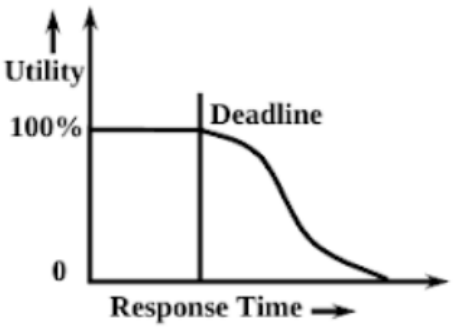

Figure 1.1: Utility of the Results Produced by a Soft Real Time Task as a Function of Time [5]

2 **Hard Real-Time System**

In hard real-time systems missing the deadline leads to a complete system failure.

The deadlines are in the order of microseconds or milliseconds.

This type of system is often safety-critical.

Examples: autopilot systems, Industrial Control Applications, and Robots.

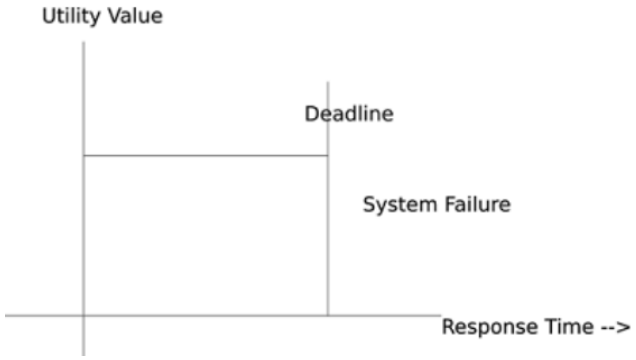

Figure 1.2: Hard Real-Time System

[5]

3 **Firm Real-Time System**

A firm real-time system is one in which a few missed deadlines will not result in total failure, but more than a few will result in complete or catastrophic system failure.

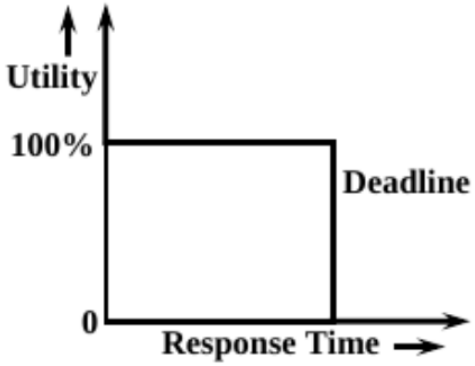

Figure 1.3: Utility of Result of a Firm Real-Time Task with Time

[5]

Every firm real-time task has a deadline by which it must complete its execution. When a firm real-time task, unlike hard real-time task, does not complete by the deadline, the system does not fail, instead, the late results are simply discarded. In other words, after the deadline, the utility of the results produced by a firm real-time task is zero. Or example : video conferencing applications

1.3.1.2 Real Time Task Model

A task model is defined by its main timing parameters The scheduling quality is based on the accuracy of these parameters, so their determination is an important aspect of real-time design.

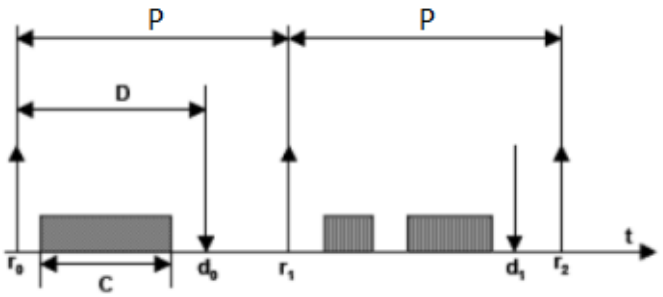

Figure 1.4: Real Time Task Model

[6]

— r, task release time, i.e. the triggering time of the task execution request.

— C , task worst-case computation time, when the processor is fully allocated to it.

— D, task relative deadline, i.e. the maximum acceptable delay for its processing.

— P, task period (valid only for periodic tasks).

When the task has hard real-time constraints, the relative deadline allows computation of the absolute deadline d = r + D. Transgression of the absolute deadline causes a timing fault.

1.3.1.3 Type Of Real-time Tasks

1 **Periodic Real-time Tasks** :

The real-time task that repeats itself after a certain time interval is called a real-time periodic task. Basically, the real-time periodic tasks are controlled by the clock interrupts. Therefore, real-time periodic tasks are also called clock-driven tasks.

For example, in a chemical plant, temperatures, pressure, and other attributes are measured periodically and all the information is transmitted to the controller.

2 **Sporadic Real-time Tasks**:

Sporadic real-time tasks are real-time tasks that occur at random intervals and have a strict deadline.

Although sporadic real-time tasks are comparable to aperiodic real-time tasks, they are not

the same.

The majority of high-critical tasks are sporadic.

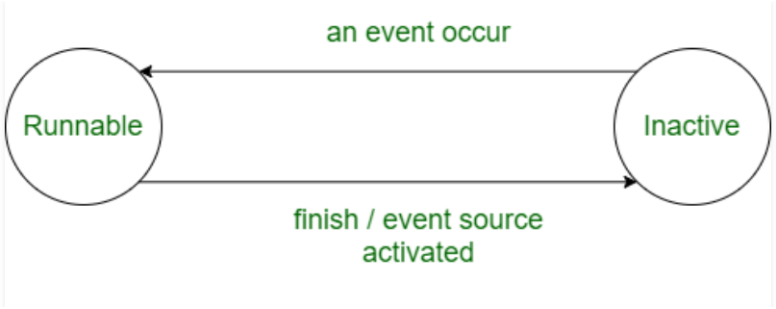

Figure 1.5: Apariodic Task

[7]

3 **Aperiodic Real-time Tasks**:

Aperiodic real-time tasks are dynamic tasks that occur at random intervals and have a soft deadline.

The time gap between two aperiodic real-time tasks could be 0. Soft real-time tasks fall within the category of aperiodic real-time tasks.

Aperiodic real-time tasks include things like typing on a keyboard and moving your mouse.

For example, typing on keyboard or mouse movements are aperiodic real-time tasks.

For example, fire handling tasks in industry or emergency message arrival in systems are sporadic real-time tasks.

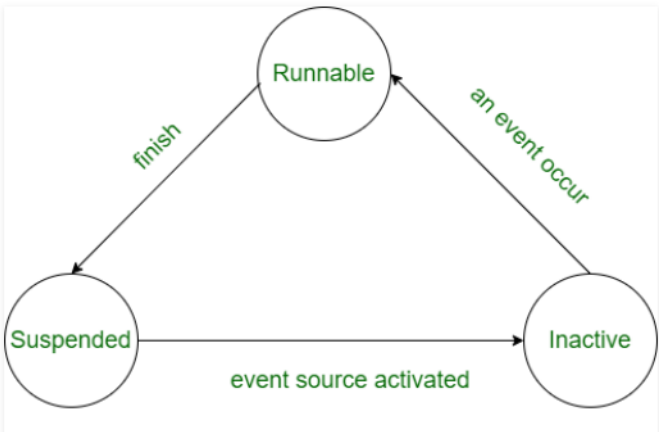

Figure 1.6: Sporadic Task

[7]

1.3.1.4 Periodic Tasks Categories

we can characterise the tasks according to the relation between its relative deadlines and the periods:

— Constrained Deadline :$T_i = D_i$ case where relative deadline is equal to the task period

— Deadline on Request: $D_i < T_i$ case where the relative deadline is inferior to the task period [8]

We can also characterise the tasks according to the relation ship between them :

— Independent.

— Dependent via resource sharing.

— Dependent through precedence relations.

1.3.2 Architecture Types

1 **Single-processor Architecture:**

A single-processor system is defined as a computer system with a single central processing unit used to perform computing tasks. As more and more modern software is capable of using multiprocessing architectures, such as SMP and MPP, the term single processor is therefore used to distinguish the class of computers where all processing tasks share a single processor. Most desktop computers have been shipping with multiprocessor architectures since the 2010s. As such, this type of system uses a type of architecture that relies on a single computing unit. All operations (additions, multiplications, etc.) are performed sequentially on the unit.

2 **Multiprocessor Architecture:**

Several forms of parallelism are exploitable in computer systems. Multiprocessor computers allow task parallelism, where one process can be executed on each processor. This provides more computing power than a single processor computer, which can be used either for several programs that would each have one processor or specially designed programs that are able to distribute their computations over the different processors.

This technology has been used for supercomputers, but it can also be used to overcome the limits of the increase in processor frequency: many current processors are said to be multi-core, and in fact, have several uniprocessors on the same chip [9].

3 **Distributed Architecture:**

Distributed architecture or distributed computing refers to an information system or network for which all available resources are not in the same place or on the same machine. This concept, of which a version can be a combination of client-server type transmissions, is opposed to that of centralized computing.

The Internet is an example of a distributed network since it has no central node. Distributed architectures are based on the possibility of using objects that run on machines distributed on the network and communicate by messages through the network [10].

1.3.3 Refactoring

It is a law of nature for fully successful iterative projects. You don't decide to refactor, you refactor because you want to do something else and refactoring helps you to do that. When you refactor existing code of a project (software/app etc) by altering its internal structure but you are not changing its external behavior.

Refactoring even take a bad design of a project and rework it into a good one. You are not changing observable behavior of the project you improves the internal structure by Refactoring.

Refactoring also removes "Code Smells" from your project, this is done to get certain benefits and these benefits may not be consumable immediately but over the long term. It is a activity which is a solution to your problems, its performed when modifying the existing code of a project to incorporate new features or to enhance.[11]

1.3.4 NP-Complete Problems

Any class of computational problems for which no efficient solution algorithm has been developed is referred to be a NP-complete issue.

This class includes many important computer-science problems, such as satisfiability problems, graph-covering problems, and the traveling salesman problem.

Solving difficult, or intractable, problems, on the other hand, necessitates times that are exponential functions of the problem size n. As the size of the problem grows larger, the execution times of the latter expand.[12]

1.4 Problematic

In this final section we'll present our problematic.

We have a set of tasks that we want to run on a set of processors. Our main goal is to find the optimal placement of these tasks that are going to be synchronised according to RM synchronization algorithm. This optimal placement must satisfy the RM feasibility condition 1.4.2.

For example let's say if our task model is composed of 3 tasks τ=$\{T_1, T_2, T_3\}$ with the following parameters :

Table 1.1: Tasks Parameters.

Tasks	$C_i(ms)$	P_i	C_i/P_i
1	4	20	0.2
2	14	30	0.46
3	4	20	0.2

and our execution platform is composed of 2 uniform processors $\mathcal{P} = \{P_1, P_2\}$.

We have 4 possible tasks placements :

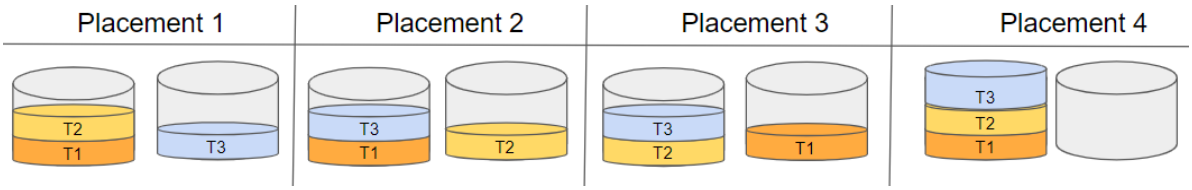

Figure 1.7: All Possible Placements

Since the processor are *homogeneous* placing T_1 in P_1 and T_2 in P_2 is equivalent to placing T_2 in P_1 and T_1 in P_2.

In fact the number of possible solutions is equal to $(m^n)/2$ where m is the number of processors and n is the number of tasks .

Table 1.2: The Processors Utilization in Every Placement.

Placement	U_{p_1}	U_{p_2}
1	0.66	0.2
2	0.4	0.46
3	0.66	0.2
4	0.86	0

Using table 1 we can conclude that placements 1, 2 and 3 are valid placements but placement 4 is not cause the processor 1 utilization U_{p_1} acceded 0.69 which means its not a feasible case. Placements 1 and 3 are the optimal placements because they have the biggest processor utilization value. So placement 1 and 3 are the solutions to this problem.

For 3 tasks and 2 processors we had 4 possible solutions, if we had 7 tasks and 4 processors the number of possible solutions is going to be $(4^7)/\ 2$ equal to 8,192. This number scales exponentially making this problem an NP-hard problem.
So in conclusion our main goal is to find the solutions of this NP-hard problem.

1.4.1 Models formalization

It is assumed in this work that the task model, that we denote by τ, is composed of n *synchronous*, *periodic*, and *independent* tasks (i.e.,$\tau = \{T_1, T_2 \dots T_n\}$). Each task T_i is characterized by static parameters $T_i = (C_i, P_i, D_i)$ where C_i is an estimation of its worst case execution time, P_i is the activation period of the task T_i, and D_i is the deadline that represents the time limit in which the task T_i must complete its execution. We assume in this work that $D_i = P_i$. The platform model, that we denote by $\mathcal{P}$, represents the execution platform of the system. We assume that this model is composed of m *homogeneous* processors (i.e., $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$). Each processor has its own memory and runs a Real-Time Operating System (RTOS). The task placement step produces a deployment model that we denote by $\mathcal{D}$ in this work. The deployment model consists of a set of tuples $\mathcal{D} = \{(P_1, \xi_1), (P_2, \xi_2), \dots (P_k, \xi_k)\}$ where k represents the number of active (or used) processors such as $k \leq m$ and ξ_j represents the subset of tasks allocated to the processor P_j after the placement step.

1.4.2 Real-time feasibility

For real-time embedded systems, the deployment model must be feasible. Feasibility means that the placement of the real-time tasks on the different processors must guarantee the respect of the timing requirements of the system. In that context, Liu and Layland [13] developed a schedulability test, which determines whether a given task set will always meet all deadlines under all release conditions. This test is based on the computation of the processor utilization factor U_p and is defined as follows:

$$U_p = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{1.1}$$

The provided schedulability test is *necessary* (i.e., if a task set is feasible, this test must be deemed *true*), but not *sufficient* (i.e., if the answer to this test is yes, this is not sufficient to say that the system is feasible). When $U_p > 1$, there is clearly no feasible schedule for the task set. In the case where the task model satisfies the RM optimal conditions [13] (i.e., tasks are periodic, synchronous, and independent, and their deadlines are equal to their periods), a *necessary* and *sufficient* feasibility test is derived for the test stated in the expression 1.1. The following is a description of the test:

$$U_p = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1) \tag{1.2}$$

We recall that

$$\lim_{x \to \infty} n(2^{\frac{1}{n}} - 1) = \ln 2 \simeq 0.69 \tag{1.3}$$

Since we assume that the task model under consideration meets the RM optimal assumptions in this study, we will assume that the deployment model is feasible if and only if

$$U_{p_j} \leq 0.69, \forall j \in \{1...k\} \tag{1.4}$$

Our problem in NP-Complete problem, in order to solve NP-complete problems many people moved toward AI , In our case we are going to use Reinforcement Learning which we will explain in detail in the next chapter.

1.5 Conclusion

Throughout this chapter we have presented the host organization, defined some necessary concepts for our work and presented the problematic. In the next chapter we will move to talk about Artificial Intelligence and it's different domains and we'll be focusing on Reinforcement Learning, its basic concepts and different algorithms.

Chapter 2

Reinforcement learning state of the art

Plan

1 Introduction 16

2 Artificial Intelligence 16

3 Machine Learning 18

4 Reinforcement Learning 21

5 Conclusion 39

2.1 Introduction

In this chapter, we will present reinforcement learning. We'll start by defining Artificial Intelligence and its different domains. Then will go over the sub-fields of machine learning. Finally will dive deep into explaining reinforcement learning, focusing in the end on Q-learning one of the reinforcement learning algorithms.

2.2 Artificial Intelligence

2.2.1 Definition Of Artificial intelligence

According to Marvin Minsky AI is " ..the science of making machines do the thing that would require intelligence if done by humans". "Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves." as stated by one of AI's pioneers, John McCarthy.[14]

So AI is a branch of computer science that focuses on simulating human cognitive thinking in machines and computer systems in other terms creating intelligent machines. These are the main aspects of intelligence:

— Generalization learning: that is learning that enables the system to adapt properly to new situations.

— Reasoning: to gather conclusions appropriate to the situation at hand.

— Problem-solving: given certain data find x.

— Perception: studying properties and interactions between objects in a scanned environment.

2.2.2 History Of Artificial intelligence

— **1943**: Warren McCulloch and Walter Pits published the first paper on artificial intelligence in 1943. They presented an artificial neuron model.

— **1949**: Donald Hebb presented a rule for changing the strength of connections between neurons this rule is known as Hebbian learning.

— **1950**: Computing Machinery and Intelligence" was published by Alan Turing proposing a test

called the Turing test that can determine the ability of a machine to be inteligent and think like a human.

— **1955**: Allen Newell and Herbert Alexander Simon created Logic Theorist the first AI program.

— **1956**: Computer scientist John McCarthy coined the term "Artificial Intelligence," at the Dartmouth Conference.

— **1966**: The creation of a chatbot by Joseph Weizenbaum called ELIZA.

— **1972**: The creation of the first intelligent robot WABOT-1 in japan.

— **1974 -1980**: The first AI winter.

— **1980**: Edward Feigenbaum presented the expert system that emulates the human decision-making ability.

— **1982**: The invention of the associative neural network by John Hopfield.

— **1986**: Navlab, the first autonomous car built by Carnegie Melon.

— **1987-1993**: Second AI winter.

— **1997**: IBM Depp Blue, a computer, defeats the world chess champion and also the first publicly available speech recognition software was developed by Dragon systems.

2.2.3 Some Sub-domains Of Artificial intelligence

These are some of AI domains:

— **Machine Learning:**

We discussed machine learning in detail in the next section 2.3

— **Natural Language Processing:**

The branch of AI that aims to make computers understand and analyze human languages .

In order to bridge the gap between human communication and computer comprehension, natural language processing depends on a variety of fields, including computer science and computational linguistics.

— **Computer Vision:**

Computer vision is a branch of AI that allows computers and systems to extract useful information from digital photos, videos, and other visual inputs, as well as to conduct actions or suggest decisions based on that data.

— **Expert System:**

Expert systems are computer programs that rely on gathering knowledge from human experts and coding that knowledge into a system.

Expert systems mimic the ability of human experts to make decisions.

Instead of using procedural code, these systems are designed to handle complex problems using bodies of knowledge.

2.3 Machine Learning

Machine Learning(ML) is a subset of AI that focuses on creating machines or programs that are capable to learn without being explicitly programmed.

These machines learn from given data sets, recognize patterns, and make judgments based on algorithms. ML algorithms are created in such a way that they can automatically learn and improve their performance.

There are three types of machine learning :

— Supervised Learning.

— Unsupervised Learning.

— Reinforcement learning.

In this section we'll go over Supervised and Unsupervised Learning and we'll go through Reinforcement learning in depth in the next section.

2.3.1 Supervised Learning

In supervised learning, the system is given labelled data sets, this data set is divided into two sets 80% as a training set and a 20% into a test set The data set consists of 2 types of variables:

— **The Target Variable:** This is what we want the machine to learn to predict, the desired output.

— **The Features:** The variables that influence the value of the target variable, the inputs we gonna feed to the machine once it learns.

For example, we want the machine to determine the price of apartments based on their surface, postal address and quality. Therefore the training data must consist of:

— prices of apartments as target data

— the surface, postal address and the quality of each apartment as features

Target Y	Features		
Price	**Surface**	**postal address**	**Quality**
313000	90	95000	3
720000	110	93000	5
250000	40	44500	4
290000	60	67000	3
190000	50	59300	3

Figure 2.1: Training data set

The machine, based on a specific algorithm, uses the training data set to create a model. A model type differs depending on the algorithms used but its main goal is to map given features to a certain prediction in our example the price of the apartment. The test data features variable is then fed to the model in order to get the predictions, these predictions are compared to the right ones and based on the results it decides how to improve the model.

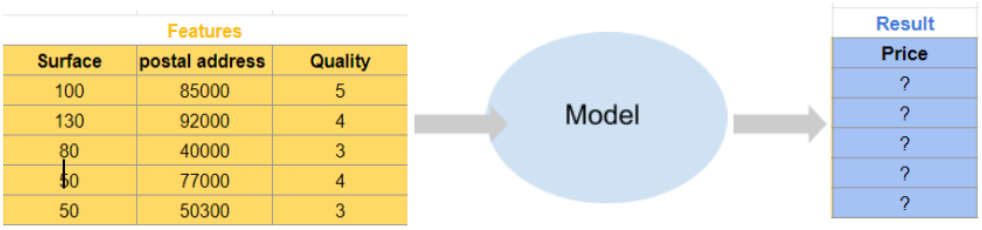

Figure 2.2: supervised learning example

Supervised learning problems are categorized into two types:

— **Classification problems:**

We use classification algorithms to Classify the discrete values such as Male or Female, Sick or not sick, True or False.

Classification algorithms :

- Random Forest
- Decision Trees
- Logistic Regression
- Support Vector Machines

— **Regression problem:**

We use regression algorithms to predict a continuous values such as:

- Predicting the age of a person.

- Predicting the price of an apartment like the example we mentioned before.

- how much will increase the stock price of a company tomorrows.

Regression Algorithms :

- Regression

- Regression Trees

- Non-Linear Regression

- Bayesian Linear Regression

- Polynomial Regression

2.3.2 Unsupervised Learning

In unsupervised learning, the system is given unlabeled data using a specific machine learning algorithm, it's able to identify hidden patterns and drive structure and classify data based on similarity.

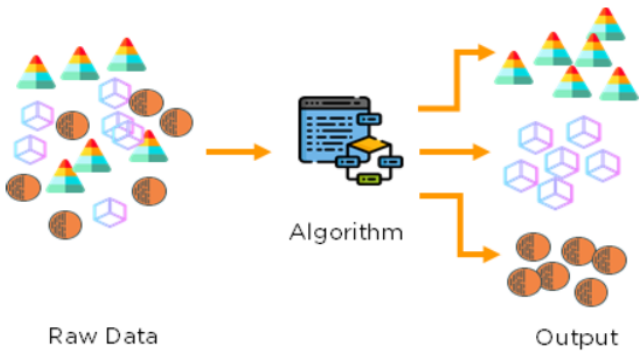

Figure 2.3: Unsupervised learning classification example

There are two types of unsupervised learning problems:

— **Clustering problems:**

It is primarily concerned with identifying a structure or pattern in a set of uncategorized data. If natural clusters (groups) exist in your data, Unsupervised Learning Clustering techniques will process them and locate them. You can also change the number of clusters your algorithms

should find.

These are some of the clustering algorithms:

- Hierarchical clustering
- K-means clustering
- Principal Component Analysis
- Singular Value Decomposition
- Independent Component Analysis

— **Association problems:**

You can create associations between data objects in huge databases using association rules. This unsupervised technique examines big databases for meaningful correlations between variables. For example, people who purchase a new home are more likely to purchase new furniture.

2.4 Reinforcement Learning

Reinforcement learning is a sub-field of machine learning that is used to train software or machines to identify the best possible behaviour in order to maximize a specific result.

Reinforcement learning is used to tackle Control and Decision tasks, in which you control a system that interacts with the real world. It is applicable in a huge number of domains like Autonomous Vehicles, Robotics, Health Care, Gaming and Resource Management.

Reinforcement learning happens through trial and error. Unlike supervised learning, the system does not have a large pre-collected data-set that it learns from it instead it learns from its own experiences interacting with the world much like humans, for example, a baby can touch fire or milk and then learns from negative or positive reinforcement, the baby takes an action,receives results from the environment about the result of that action, repeats this process till it learns which actions are good and which actions are not[15].

To apply reinforcement learning, the first step is to formalize the problem as a Markov Decision Process (MDP).

2.4.1 Markov Decision Process

Markov Decision Process has five components:

— The agent is the system or machine you operate.

— The environment that the agent interacts with.

— The environment states.

— The actions that the agent can take in order to interact with the environment.

— The reward that represents the outcome of the action can be positive or negative. It's how the agent evaluates the goodness or badness of its behaviours. This is how the agent learns from experience so it is crucial that we define the rewards in a way that truly reflects the behaviour that we want our agent to learn.

The agent interacts with the environment over a sequence of time steps.

If we have a set of states S, actions A and a set of rewards R, assuming that each of these sets has a finite number of elements.

At each time step t, the agent will get some representation of the environment's state S_t in S, given this representation, the agent selects an action to take A_t in A, time is then incremented to the next time step $t+1$ the environment is then transitioned to some new state S_{t+1} and the agent is given a reward as a consequence of its previous action. R_{t+1} in R.

We can think of the process as an arbitrary function f that maps state-action pairs to rewards at each time t $f(S_t, A_t) = R_{t+1}$.

This diagram illustrate this entire process 2.4.

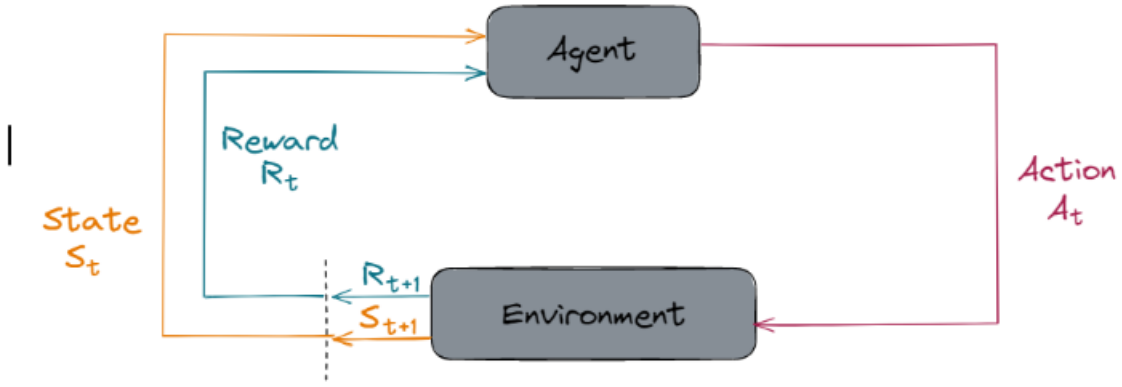

Figure 2.4: RL Diagram

This execution of MDP can be represented using a trajectory that shows the sequence of

state actions and rewards. The trajectory can be represented like this:

S_0 ,A_0 , R_1 , S_1 , A_1 , R_2 , S_2 , A_2 , R_3

The agent's purpose throughout the procedure is to maximize the total amount of rewards it gets from taking action. This means the agent wants to maximize both the immediate and cumulative rewards 2.4.3.1 it will get overtime.

2.4.2 Episodic and Continuous Tasks:

When a task sequence has a well-defined start(start time step) and a well-defined end (final time step) it's known as an episode. For example, playing a game of tic-tac-toe, every new round of the game represents an episode.

When a player scores a point the episode's last time step takes place. At the end of the episode the starting state is either reset or randomly chosen from the possible environment states.

We can represent episodic tasks as so :

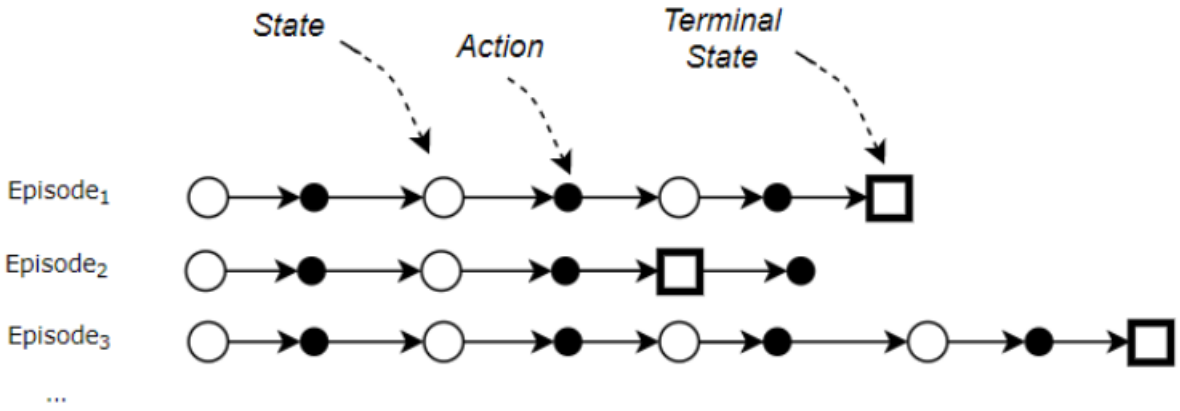

Figure 2.5: episodic tasks

Every episode is independent of the other once.

On the other hand, tasks that have no final step(end) and can go forever are called Continuing Tasks, for example, a personal assistance robot.

2.4.3 How to solve a reinforcement learning problem?

In order to solve a reinforcement learning problem we need to get these major concepts at first:

2.4.3.1 The Return

Return is the cumulative reward meaning some of reward from the first time step till the final time step, it is the sum of future rewards.
The return G at time t is defined mathematically as

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

T is the Final time step.
This concept is hugely important because the agent's goal is to maximize the expected return, the agent's decision-making is driven by the predicted return.

To insure that the cumulative rewards do not continue to grow indefinitely as the number of time-steps grows (like for continuing tasks, or for very long episodes) we apply a discount factor γ, a number between 0 and 1 to weight future rewards.
The discounted reward is defined like this :

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \end{aligned}$$

It also pushes the agent to prioritize the immediate reward above later rewards, as the latter will be greatly discounted.

2.4.3.2 Policy

Policy is any strategy used by the agent to determine which action to take in a certain state. A policy answer the question how probable is it for an agent to pick any action from a given state? A policy maps a state to an action, it can be a look-up table or a function.
In this example 2.6 the a gent in state 3 chose action a_2 by consulting the table to determine which action it should take.

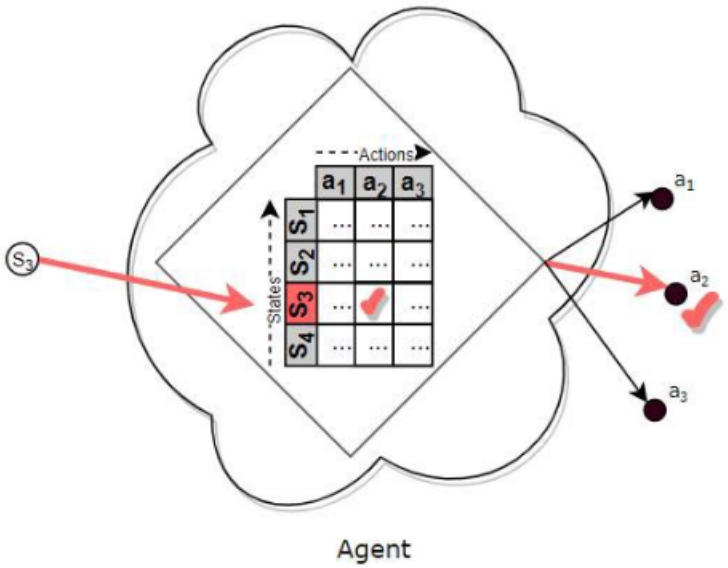

Figure 2.6: example 1

For example if the agent follows policy π at time t then π(a|s) is the probability that the agent will choose action a in state s under policy π . There are two types of policies :

- Deterministic : The agent choose the same action at a certain state.

- Stochastic: The agent picks the action for every state based on the probability of each action.

This example illustrate the two different types :

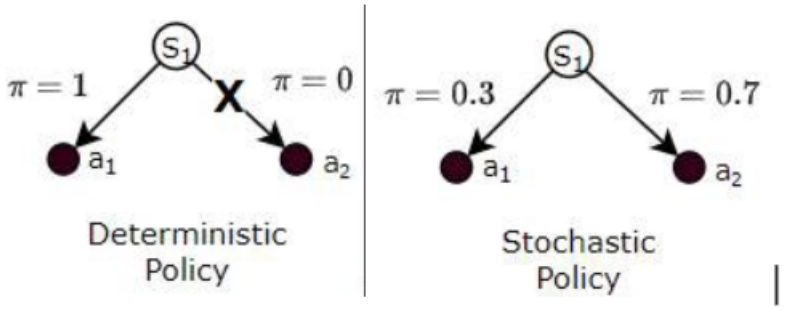

Figure 2.7: Policy Types

When the agent first starts off, it doesn't have a useful policy and has no clue what action it should perform in any given state. Then, using the Reinforcement Learning method, it gradually learns a good policy.

2.4.3.3 Value Functions:

There are two types of value functions:

— State Value Functions.

— State Action Value Functions.

Value functions determine how good is any given state or any given action in a certain state for an agent.

This idea of how good a state our state-action pair is dependant on the expected return.

As we mentioned earlier the rewards an agent expects to receive are dependent on what actions the agent takes in given states 2.4.1.

Since the agent chooses its action according to a policy 2.4.3.2then we can conclude that value functions are defined with respect to policies.

State-Value Function:

The state-value function for policy π, denoted as v_π , gives us the value of a state under π. Using this value we can tell how good any given state is for an agent following policy π.

let's say if the agent has been given a policy π and it's in state s at time t.

The state value of s following policy π is the average return the agent can get if it starts from s and always picks actions based on policy π [15].

$v_\pi(s)$ is defended as follow

$$\begin{aligned} v_\pi(s) &= E_\pi\left[G_t \mid S_t = s\right] \\ &= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]. \end{aligned}$$

Action-Value Function:

Similarly, the action-value function for policy π, denoted as q_π gives us the value of an action in a given state under π. Using this value we can tell how good any given action in any given state following policy π.

The value of action a in state s under policy π is the expected return from starting from state s at time t, taking action a , and following policy π thereafter. We define $q_\pi(a,s)$ as

$$q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a]$$
$$= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right].$$

2.4.3.4 Optimality

The agents goal is to maximize his return therefore he must find the Policy that gets him the highest return than all other policies. This policy is called the optimal policy.

Optimal Policy:

A policy π is considered to be better than π_1 or the same as policy π_1 if the expected return of π is greater than or equal to the expected return of π_1 for all states. In other words a policy that is better than all other policies is called the optimal policy [16].

$$\pi \geq \pi' \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in \boldsymbol{S}.$$

Optimal State-Value Function:

The optimal policy has an optimal state-value function. We denote the optimal state-value function as v_* and define as

$$v_*(s) = \max_\pi v_\pi(s) \quad \text{for all } s \in \boldsymbol{S}.$$

In other words, v_* gives the largest expected return achievable by any policy π for each state.

Optimal State-Value Function:

The optimal policy has an optimal state-value function.We denote the optimal state-value function as q_* and define as

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad \text{for all } s \in \boldsymbol{S} \text{ and } a \in \boldsymbol{A}(s).$$

In other words, q_* gives the largest expected return achievable by any policy π for each possible state-action pair.

q_* must satisfy the Bellman Optimality Equation :

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

It states that, for any state-action pair (s,a) at time t the expected return from starting in state s, selecting action a and following the optimal policy thereafter (the Q-value of this pair) is going to be the expected reward we get from taking action a in state s , which is R_{t+1}, plus the maximum expected discounted return that can be achieved ($s^{'}$,$a^{'}$).
Since the agent is following an optimal policy, the following state ($s^{'}$) will be the state from which the best possible next action ($a^{'}$) can be taken at time $t+1$ [15].

As a conclusion of this section solving a reinforcement learning problem consist of structuring the problem as MDP, creating an agent model and training it to find the optimal policy by applying reinforcement learning algorithms.
So solving a reinforcement learning problem consists of finding the optimal policy.

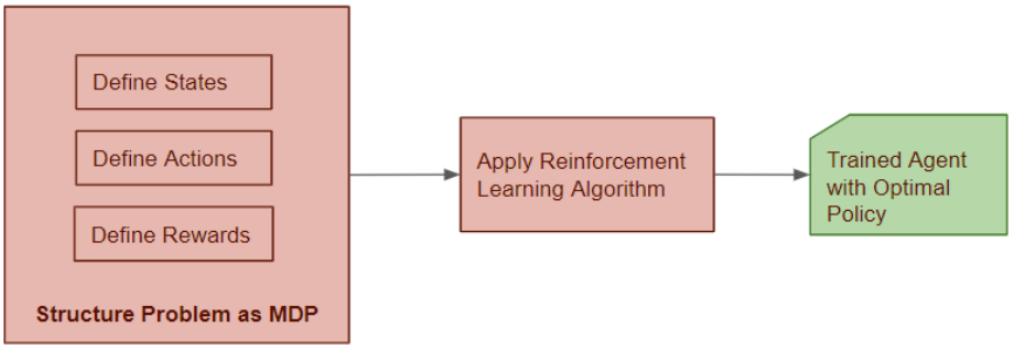

Figure 2.8: Solving a Reinforcement learning problem approach

2.4.4 Reinforcement learning Algorithms:

There are many RL problems that we can group into different categories.

Model Based and Model Free Approaches:

Model-based approaches are used when the environment's internal operation is known When

any Action is performed from some Current State, we can predict with certainty what Next State and Reward will be output by the environment like illustrated in this example:

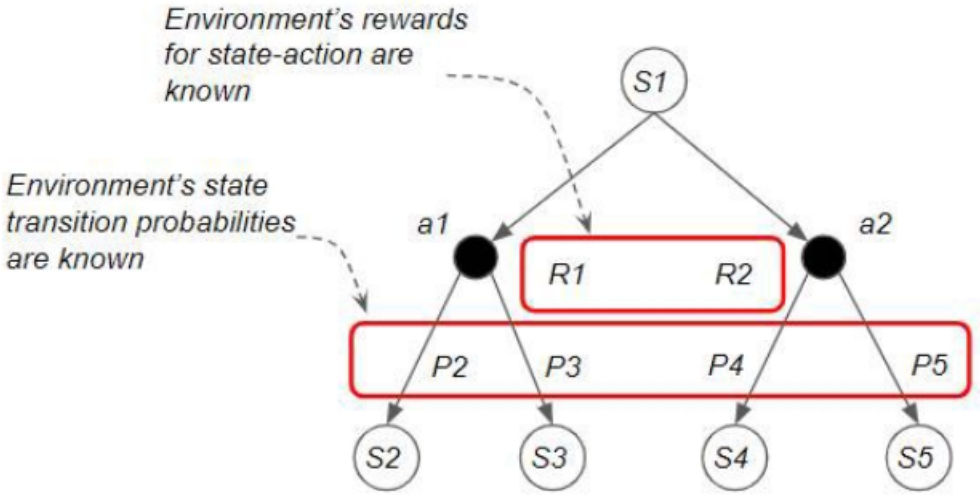

Figure 2.9: Model-based approach

where S_1, S_2, S_3, S_4, S_5 are states and a_1 a_2 are actions. The agent is in state S_1 and can take ether action a_1 or a_2. If the agent chose action a_1 there is a P_2 probability that he will transition to state S_2 and a P_3 probability that he will transition to state S_3 and if he chose action a_2 there is a P_4 probability that he will transition to state S_4 and a P_5 probability that he will transition to state S_5.

Model-free approaches are used when the environment is highly complicated and its internal dynamics are unknown as shown in this figure2.10.
We consider the environment as a black -box.

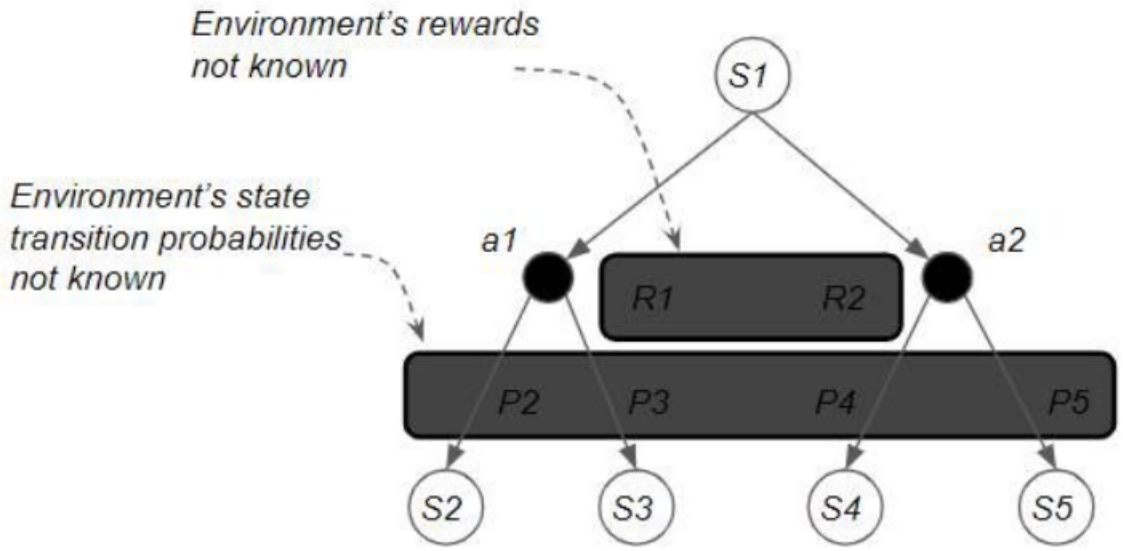

Figure 2.10: Model-free approach

Prediction vs Control problems :

In prediction problem policy is given as input and the goal is output the corresponding Value function.

In control problems no policy is provided with the goal is to examine all the possible policies to find the best one.Control problems are the most practical ones since the main goal is to find the optimal policy.

Most interesting real-world problems are Model Free Control problems. This is what most people mean when they use the term Reinforcement learning [17].

These are the most common RL algorithms

	Prediction	Control
Model-based Planning	• Dynamic Programming Policy Evaluation	• Dynamic Programming Value Iteration • Dynamic Programming Policy Iteration
Model-free Reinforcement Learning	• Monte Carlo Prediction • Temporal Difference TD(0) • TD(λ) Backward	• Monte Carlo Control • Sarsa • Sarsa Backward • Q Learning • Deep Q Networks • Policy Gradient • Actor Critic

Figure 2.11: Reinforcement learning algorithms

2.4.5 Model-free algorithms

To solve control problems we can use two types of algorithms :

— policy-based algorithms: find the optimal policy directly.

— State-Action Value-based algorithms: find the optimal state-action function and then find the optimal policy .

To find the optimal policy from the optimal state-action function we just pick the action with the highest state-action value.

The optimal policy is deterministic in general because it always selects the best action but if there is a tie between the two Q-values, the Optimal Policy could be stochastic.

In that instance, the Optimal Policy chooses with equal probability one of the two corresponding actions. This is usually the case when the agent plays a game against an opponent. Because a deterministic policy would result in the agent making predictable moves that its opponent could easily beat, a stochastic Optimal Policy is required.

To solve prediction problems we use State-Value-based algorithms:

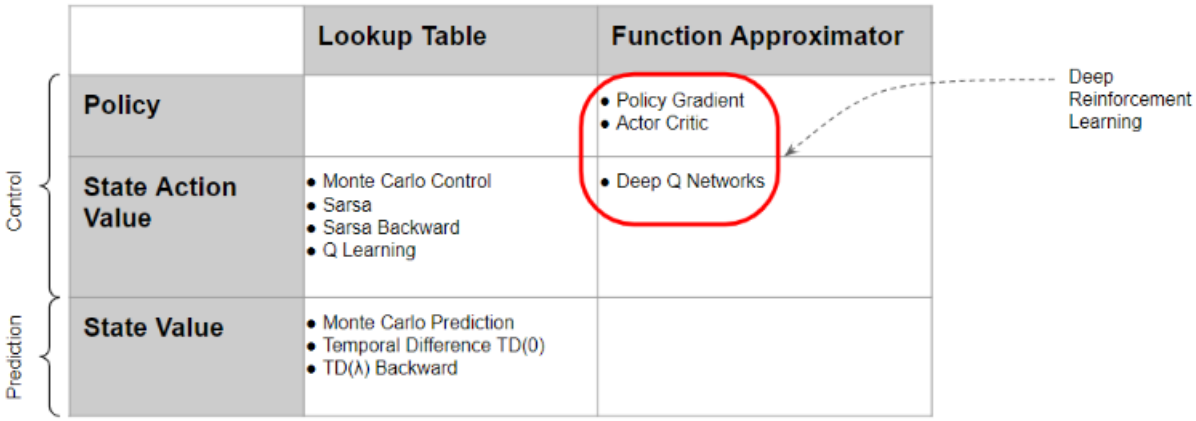

Figure 2.12: model free algorithms.

All model-free algorithms both State-Value-based and Policy-based are based on iterative solutions.

2.4.6 Q-learning

Q-learning Objective is to find the optimal policy the one that yields the maximum achievable return. In order to find this policy Q-learning starts by learning the optimal Q-values for each state-action pair and can conclude the optimal policy using it.

The Q-learning is an iterative algorithm, in this algorithm Q-values for each state-action pair are updated using the Bellman equation until the Q-function converges to the optimal Q-function q_*. This is called value iteration.

In order to explain this algorithm in detail we are going to set up an example using a Mario Game.

Suppose we have the following environment. The agent in our environment is Mario. Mario wants to get to the princess and rescue her in the least amount of time without stumbling on lava pits, that can kill him.

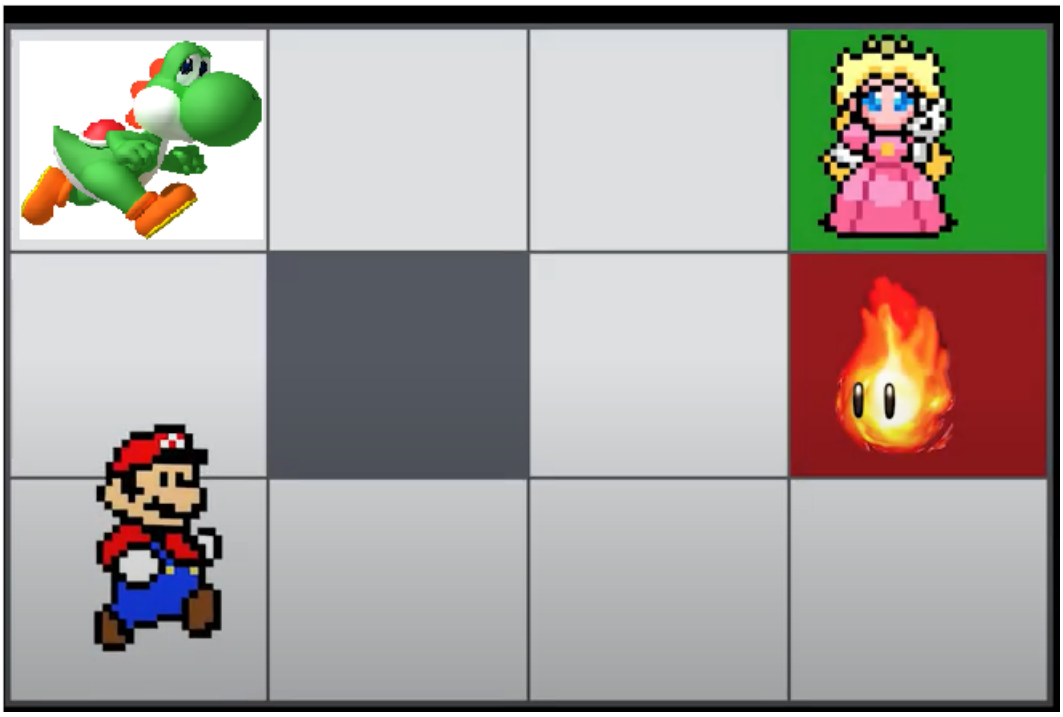

Figure 2.13: Mario Game.

Mario can move left, right, up, or down in this environment. These are the actions. The states are determined by the individual tiles and where Mario is on the board at any given time.

-If Mario lands on an empty tile the reward is minus 1 point.

- A tile with the dinosaur is plus 3 points and will end the episode.

- A tile with the princess is plus 10 points and will end the episode.

- A tile with lava is minus 10 points and will also end the episode.

State	Reward
Empty	-1
Dinosaur	+3
Princess	+10
Lava	-10

Figure 2.14: Game Rewards.

At the start of the game, Mario has no idea how good any given action is in any given state. he is only aware of the current state of the environment. In other words, it doesn't know from the start what move to take, left, right, up, or down and if that action will yield a positive reward

or negative reward. Since Mario, knows nothing about the environment all the Q-values for each state-action pair will be initialized to zero these Q-values will be iteratively updated using value iteration Throughout the game.

We'll be making use of Q-table, to store the Q-values for each state-action pair. The horizontal axis of the table represents the actions, and the vertical axis represents the states.

As the Q-table becomes updated, in later moves and later episodes, Mario can look in the Q-table and base its next action on the highest Q-value for the current state.

Now, we'll set some standard number of episodes that we want Mario to play. Let's say we want him to play six episodes. During these six episodes that the learning process will take place.

In each episode, Mario starts out by choosing an action from the starting state based on the current Q-values in the table.

Mario chooses the action based on which one has the highest Q-value in the Q-table for the current state.

But, all the Q-values are set zero at the start, so there's no way for Mario to differentiate between them to discover which one is considered better. So, what action does it start with?
To answer this question we need to introduce tow big terms exploration and exploitation. This will help us understand how exactly the agent chooses actions in the entire game.

2.4.6.1 Exploration and Exploitation

Exploration is the act of exploring the environment in order to discover all possible paths to know which one is the best. Collecting all possible information about the environment.
Exploitation is the act of using the information that is already collected about the environment, in the exploration phase, to maximize the return.

Our agent wants to maximize the expected return, so you might think that we want our agent to use exploitation all the time and not worry about doing any exploration. This strategy isn't quite right. Think of our game. If Mario got to the Dinosaur before it got to the Princess, then only making use of exploitation, going forward Mario would just learn to exploit the information it knows about the location of the Dinosaur to get 3 incremental points infinitely.
If Mario was able to explore the environment, however, he would have the opportunity to find

the Princess and get a bigger reward. If Mario only explored the environment with no exploitation, however, then it would miss out on making use of known information that could help to maximize the return.

The challenge consists of finding the right balance between exploration and exploitation . To get this balance we will use what is called an epsilon greedy strategy.

2.4.6.2 Epsilon Greedy Strategy

Epsilon greedy strategy is based on an exploration rate ϵ that we initialized to 1. This exploration rate represents the probability of exploring the environment rather than exploiting it. It is 100% certain that the agent will start out by exploring the environment since we initialized ϵ to 1.

At the start of each new episode, we will decay ϵ by some rate that we set so that exploration becomes less and less probable as the agent learns more and more about the environment. We say the agent will become "greedy" in terms of exploiting the environment.
So as training progresses the ϵ is updated to ensure more exploration in the early stages and a transition to more exploitation in the later stages like shown in this figure 2.15

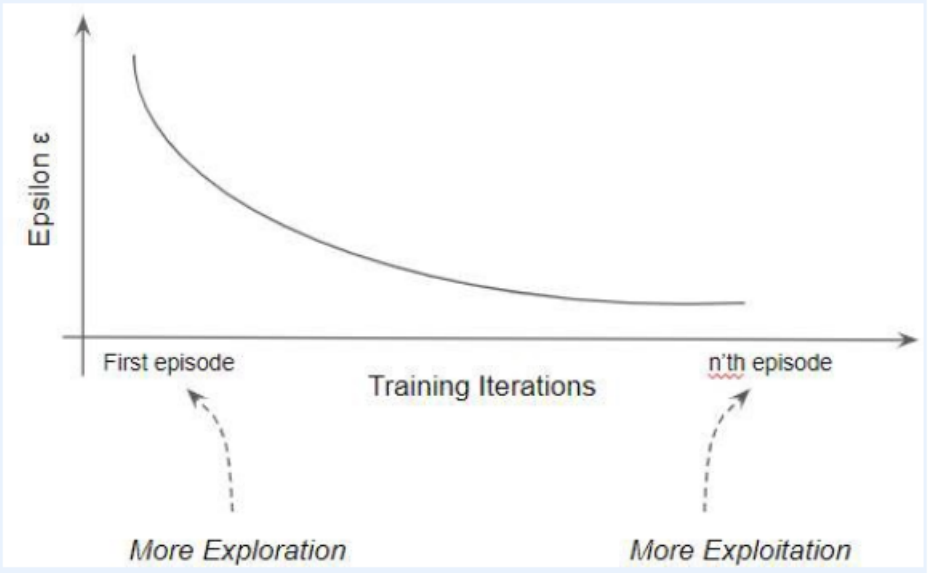

Figure 2.15: Exploration and Exploitation with epsilon-greedy

To determine if the agent will choose exploration or exploitation at each time step, we generate a random number between 0 and 1.If this number is greater than epsilon,the agent will choose its next action via exploitation,it will choose the action with the highest Q-value for its current state from the Q-table.If it is less the action will be chosen via exploration, randomly chosen to explore

the environment.

2.4.6.3 Choosing An Action

So We started talking about the exploration-exploitation because we were discussing how Mario should choose the very first action since all the actions have a Q-value of 0.

Now, we know that the action will be chosen randomly via exploration since our exploration rate is set initially to 1. Meaning, that with a 100% probability, Mario will explore the environment during the first episode of the game, rather than exploit it.

So after Mario takes an action, it observes the next state, the reward gained from its action, and updates the Q-value in the Q-table for the action it took from the previous state.

Let's suppose the Mario chooses to move right from the starting state.The reward we get for taking this action is -1 cause Mario moved to an empty tile and empty tiles have a reward of -1 point.

2.4.6.4 Updating The Q-Value

To update the Q-value for the action of moving right taken from the initial state, we use the Bellman equation we mentioned earlier.

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

Our goal is to make the Q-value for the given (a,s) pair as close as possible to the right-hand side of the Bellman equation so that the Q-value will converge to the optimal Q-value q_*.
We do this by comparing the loss between the Q-value and the optimal Q-value for the given state-action pair and then updating the Q-value at each iteration to reduce the loss.

$$q_*(s,a) - q(s,a) = loss$$

$$E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right] - E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] = loss$$

We first need to introduce the idea of a learning rate to actually show how we update the Q-value.

2.4.6.5 Learning Rate

Learning rate is a number between 0 and 1, it represents how quickly the agent update the previous Q-value in the Q-table for a given state-action pair for the new Q-value.
For example, suppose we have a Q-value for some arbitrary state-action pair, the agent has already experienced this pair in a previous time step, if the agent experiences that same state-action pair at another time step that Q-value will need to be updated in order to change the expectations the agent has for the future returns [17].

We don't just overwrite the old Q-value, but rather, we use the learning rate to determine how much information we keep about the previously computed Q-value for the given state-action pair versus the new Q-value calculated for the same state-action pair.

We denote the learning rate with the symbol α , and we'll arbitrarily set α to 0.7 for our game example.

The agent will adopt the new Q-value more quickly if the learning rate is higher. So for example, if the learning rate is 1, the estimate for the Q-value for a given state-action pair would be the straight-up newly calculated Q-value and would ignore previous Q-values.

2.4.6.6 Calculating The New Q-Value

This is the equation for calculating the new Q-value for the current state-action pair(s,a) , where s is the initial state an a is the action of moving right:

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\Big(\overbrace{R_{t+1} + \gamma \max_{a'} q(s',a')}^{\text{learned value}}\Big)$$

So new Q-value is equal to a weighted sum of old Q value and the learned value.

The old value in our case is 0 since this is the first time the agent is experimenting this state-action pair, we multiply this old Q-value by (1-α).

Our learned value is the reward the agent receives plus the discounted estimate of the optimal future Q-value for the next state-action pair ($s^{'}$, $a^{'}$) a time $t+1$. This entire learned value is then multiplied by the learning rate.

Suppose the discount rate $\gamma = 0{,}9$. This is how we update the Q-value of our (s,a) pair.

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\Big(\overbrace{R_{t+1} + \gamma \max_{a'} q(s',a')}^{\text{new value}}\Big)$$
$$= (1-0.7)(0) + 0.7\left(-1 + 0.99\left(\max_{a'} q(s',a')\right)\right)$$

Let's focus on the term $max_{a'}$q($s^{'}$, $a^{'}$). All the Q-values are initialized to 0 in the Q-table at the current time, we have

$$\begin{aligned}\max_{a'} q(s',a') &= \max(q(\text{empty6, left}), q(\text{empty6, right}), q(\text{empty6, up}), q(\text{empty6, down}))\\ &= \max(0,0,0,0)\\ &= 0\end{aligned}$$

Now, we can now substitute the value 0 in for $max_{a'}$q($s^{'}$, $a^{'}$) to solve our equation : .

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\Big(\overbrace{R_{t+1} + \gamma \max_{a'} q(s',a')}^{\text{new value}}\Big)$$
$$= (1-0.7)(0) + 0.7\left(-1 + 0.99\left(\max_{a'} q(s',a')\right)\right)$$
$$= (1-0.7)(0) + 0.7(-1 + 0.99(0))$$
$$= 0 + 0.7(-1)$$
$$= -0.7$$

finally we'll take this new Q-value and store it in the Q-table for this particular state-action pair.

We've completed all of the steps required for a single time step. This process will repeat itself for each time step until the episode comes to an end.

We will have our optimal policy after the Q-function converges to the optimal Q-function.

2.4.6.7 Max Steps

We can also set a limit on how many steps our agent can take before the episode ends automatically. With the current setup of the game, it will only be terminated if Mario reaches the

state with dinosaur , princess , or the state with lava.

If the Mario hasn't reached termination via either of these three states after the 100th step, we may define a condition that says the game should end after the 100th step.

As a summary these are all of the steps for Q-learning we went through :

Initialize all Q-values to 0 in the Q-table.

In each episode for each time-step :

1- select an action using ϵ greedy.

2- Observe the reward and next state.

3- Update the Q-value using bellman's Equation.

2.5 Conclusion

In this chapter, we covered the domain of Artificial Intelligence and how it is changing technology to be aware and smart just like humans. We then presented Reinforcement Learning a sub-field of machine learning which it self is a sub-field of Artificial intelligence. Where we talked about machines that are able to learn by interacting with the world and we explained in detail the Q-learning algorithm which is the basis of our solution that we'll discuss in the next chapter.

Chapter 3

Reinforcement Learning Model for The Task Placement Problem

Plan

1 Introduction 41

2 Q-learning Model 41

3 Algorithms 43

4 Conclusion 54

3.1 Introduction

In this final chapter, we'll start by defining our solution based on a Q-learning model. After that, we'll analyze the results of a specific case study and we'll see how our model can be used for refactoring.

3.2 Q-learning Model

In this section, we provide a model based on Q-learning for finding the optimal solution to a placement problem in a distributed real-time system. The optimal solution is one that ensures task placement with respect to their deadline constraints while reducing the number of active processors. As already explained, we consider a platform model with m processors and n tasks to be mapped to the minimum number of processors. This problem is assumed to be NP-hard since the number of placement solutions (combinations) is large, but few of them are feasible. Q-learning is ideally suited to challenges in which only a portion of the environment is observed as is the case here. To adapt Q-learning with the aim of resolving task placement problem for real-time distributed systems, some RL notions are redefined as the following:

State : At time step t, a state S_t is represented by the collection of tasks already placed on the processors with respect to their deadlines (i.e., D_t a deployment model at time step t), as well as the list L_t of tasks that have not yet been placed.

Action space : The action space represents the set of possible actions that the agent can take when it encounters a state S_t following a policy π. In our case, the action is to find the best processor for a given task from the list of unplaced tasks, following an ϵ-greedy policy

Reward : In RL methods, the reward is a crucial key. As a result, deciding on the rewards and/or penalties requires great attention. The agent is reward-motivated throughout the process, and he is learning how to maximize it through trial and error experiences in the environment. We define the reward to teach the agent how to act in various situations. Hence, the following points must be considered:

— The agent should receive a high positive reward for optimal placement because this behavior is highly desired

— The agent should be penalized for prohibited placement (if it tries to place a task on a full processor)

— The agent should get a slight negative reward for not making an optimal placement. Slight negative because we would prefer that our agent make a sub-optimal placement rather than make a prohibited placement (placement on full processor)

Because our goal is to reduce the number of active processors, a new variable E_j to indicate the processor state is required. In fact, when the agent selects a new processor P_j which is in sleep mode, E_j equals -1; otherwise, E_j equals 0. The agent is penalized by the negative value of E_j if it tries to wake up a new processor. The goal is to prioritize task placement on currently active processors, with sleep processors being used only when all active processors are completely utilized. Another case to note is when the agent tries a prohibited placement; in this case, E_j is subjected to a large negative penalty, and so E_j is equal to $-m$. (the maximum number of processors). The total reward R obtained at the end of a task placement is given by

$$R = E_j - U_{jt} + U_i \tag{3.1}$$

Where

— U_{jt} is the available utilization of processor P_j at time step t

— U_i is the required utilization for a task T_i (i.e., $U_i = \frac{C_i}{P_i}$)

— E_j reflects the processor state such as

$$E_j = \begin{cases} \text{-1} \\ \text{-m} \\ 0 \quad \text{otherwise} \end{cases}$$

Policy : In this work we use the ϵ-greedy policy, which is the most well-known strategy for Q-learning. Indeed, this policy permits balancing exploration (the selection of a random action) and exploitation (the choice of actions is based on already learned Q-values). We shall decrease ϵ from one episode to another to favor exploitation and so profit from the agent's previous trials learning.

Epoch : The decision epoch matches the placement of all the tasks; it operates on sequential steps. At each step, the agent observes the environment state and searches for the optimal pair (i.e., (processor*,task*)).

Q-table : The Q-table represents a matrix, where rows represent all possible states and columns refer to all actions. Firstly, it is initialized to 0, and then values (denoted as $Q(S, a)$) are updated at each step of an epoch. The generation of the Q-table that defines all the possible states for a given application is only performed for the initial deployment.

3.3 Algorithms

Algorithm 1 describes the proposed task placement solution for real-time distributed systems based on the RL approach. The proposed algorithm has some initialization parameters, such as α, which represents the learning rate (we assume $\alpha = 0.5$), and γ, which represents the discount factor that quantifies the importance given to future rewards (we assume $\gamma = 0.9$).

Input: $\alpha = 0.5$

$\gamma = 0.9$

$\epsilon = 0.9$

ϵ_decease_factor= 0.001

L_t: List of unplaced tasks

$\mathcal{P}$: List of processors

Output: $\mathcal{D}*$: Optimal deployment model

Notations:

Q : The Q-table

$\mathcal{D}_t$: The deployment model at time step t (i.e., list of (processor, tasks) pairs)

S: The state at the time step t (i.e., $S \leftarrow (L_t, D_t)$)

Generation of Q *[r]only performed for the initial placement *Initialization of* $Q(S, a)$ *to* *0 , for all* S *and* a;

while $Q(S, a)$ *still moving* **do**

reset S;

while L_t *is not empty* **do**

$a \leftarrow$ Select_Placement (ϵ, L_t, $\mathcal{D}_t$);

Take_Action a then Compute (R); *[r]compute reward using expression **??**

Observe $Q(S\prime, a\prime)$;

$Q(S, a) = Q(S, a) + \alpha[R + \gamma max_{a\prime} Q(S\prime, a\prime) - Q(S, a)]$; *[r]Bellman's equation

Update $Q(S, a)$;

Remove (T_i, L_t); *[r]remove T_i from L_t $D_{t+1} \leftarrow$ Update ($\mathcal{D}_t$); *[r]place T_i on P_j

$S \leftarrow S'$;

end

$\epsilon \leftarrow max(\epsilon$ - ϵ_decease_factor, 0);

end

return $\mathcal{D}*$;

Algorithm 1: Task Placement with RL

The list of unplaced tasks (L_t) and the list of processors ($\mathcal{P}$) are also given as inputs. Because the ϵ-greedy is used as a policy in this model, the ϵ value must also be defined. Indeed, the ϵ value is decreased over each epoch by ϵ_decease_ factor. This ϵ degradation can be explained by the fact that agent learning grows from one epoch to the next, and new states are visited as the process runs,

so we prefer exploitation over exploration. The output of this algorithm is an optimal deployment model which corresponds to the best placement of the input tasks on the given processors. As already mentioned, the best placement is the one that guarantees the real-time feasibility of the system while minimizing the number of used processors. In order to hold the values of all potential state-action pairings, Algorithm 1 first constructs the Q-table Q (based on L_t and $\mathcal{P}$). This generation occurs only on the first deployment of a certain application.

Input: ϵ: The epsilon value

$\mathcal{D}_t$: The deployment model of the current state

L_t: List of unplaced tasks in the current state

Output: a: placement to be done

Notations: Q : The Q-table

$nb \leftarrow random(0, 1)$ *[r]generate a uniform number between 0 and 1 **if** $nb \leq \epsilon$ **then**

  $T_i \leftarrow$ random task from L_t;

  $P_j \leftarrow$ random processor from $\mathcal{D}_t$;

  $a \leftarrow (P_j, T_i)$;

else

  Select a from Q where Q(S,a) = max Q(S,a);

end

return a;

Algorithm 2: Select_Placement

3.3.1 Case study

In this section, we consider a case study dealing with a system embedded in a car that displays various data to the driver. The specified system's functionalities are ensured by five tasks; three sensing tasks and two display ones. The platform model consists of three homogeneous processors that we denote by P1, P2 and P3. The application tasks are assumed to be independent and periodic, with an execution time of C_i in the worst-case scenario and a period of P_i equal to the deadline D_i. Table 3.1 gives a tabular description of the task model describing the considered case study.

Table 3.1: Task model tabular description of the considered case study

Task	Description	C_i (ms)	P_i (ms)	U_i
SENSOR_1	This task reads the car's speed	1	10	0.1
SENSOR_2	This task provides information about the temperature of the car	2	10	0.2
SENSOR_3	This task reads the GPS position of the car	3.2	40	0.08
DISPLAY_1	This task displays a summary of the sensing data	2	12	0.16
DISPLAY_2	This task displays the map of the current car location	1	6	0.16

Now our case study is ready and its time to apply our RL based model on. As a first step, we begin by defining the driven system as a MDP.

It is clear that the environment is the set of the five tasks (L) that we have to map to at most 3 initially unused processors. An environment state S_t at the step time t is represented by the set of tasks not yet received by any processor and the set of processors with their corresponding actions already on. An action corresponds to finding a pair (processor index, task index). A state transition occurs when the agent executes an action and as a consequence the state S_t evolves to S_{t+1} as a result. Finally the reward is computed as previously explained. The application of Algorithm 1 results first, in the generation of all the possible states then the creation of the Q-table followed by a Q-values zero initialization. After that the process of actions selection and states transition begin until a great number of Q-values is updated (the agent arrives at visiting the majority of states). Thus the Q-table is ready to be exploited by the agent that has to select, at each transition, the pair state-action (S-a) with the highest Q(S,a) value until all the tasks are mapped to the set of the given processors. The deplyment model resulting as the application of this lately described process to our case study is as shown by this table 3.2.

Table 3.2: Deployment Model.

Tasks	Processor	Processor utilization
SENSOR-2	P1	$U_{p_1} = 0.2; U_{p_2} = 0; U_{p_3} = 0$
DISPLAY-1	P1	$U_{p_1} = 0.36; U_{p_2} = 0; U_{p_3} = 0$
DISPLAY-2	P1	$U_{p_1} = 0.52; U_{p_2} = 0; U_{p_3} = 0$
SENSOR-1	P1	$U_{p_1} = 0.62.; U_{p_2} = 0; U_{p_3} = 0$
SENSOR-3	P2	$U_{p_1} = 0.62; U_{p_2} = 0.08; U_{p_3} = 0$

As we can note the agent is not allowed to place a task T_i on a new processor p_j only if the previous ones (processors with index $k<j$) are unable to host T_i ($U_{T_i}<U_k$ for all $k < j$). This result is one of the advantages of the integration of RL in placement approach that leads to smart tasks placement that is a placement solution with an economic number of processors that ensures the correct functioning of the system thus processor can be put on sleep mode. To execute the process, algorithms are implemented in Python3 with NumPy library and we use a PC equipped with I72700H intel processor and 16GB RAM The total execution time is equal to 3.39s shared between generation of the set of all tasks which is equal to 2.1 s to be done and the time of the filling of Q-table values and the searching for the optimal solution estimated to be 1.29s.

With help of this experiments it is proved that the majority of the execution time id dedicated to the generation of the states.

Now suppose that for special reason the designer of the car detects that the frequency in witch the system detects and sends information to the driver are very late and thus decides to reduce the periods to the half for all the case study tasks. The process of features tasks or processors update is called refactoring and it is very known in real time applications. Table 3.3 presents updated features tasks.

Table 3.3: Tasks new features.

Tasks	$C_i(ms)$	P_i	C_i/P_i
SENSOR-1	1	5	0.2
SENSOR-2	2	5	0.4
SENSOR-3	3.2	20	0.16
DISPLAY-1	2	6	0.32
DISPLAY-2	1	3	0. 32

Our solution is composed of two major parts :

1 States generation (witch takes the majority of the spent process execution time).

2 Filling the Q-table and finding the optimal solution.

state generation is independent of tasks and processors features and it only depends on their numbers. So for two different cases with the same number of tasks and number of processor our solution gonna take the same time to generate all the states and these sates gonna be the same for both cases.

Thus to find the deployment model for this second example we only need to compute the new Q-values then exploit them to obtain the optimal placement making the execution time to generate the new deployment model is 1.29s.

This solution feature allow us to win a lot of time in comparison to the ordinary application where the small change introduced in tasks period leads to the redone of the the placement process from scratch. Table 3.4 shows the new deployment model for the updated case study.

Table 3.4: Deployment Model.

Tasks	Processor	Processor utilization
DISPLAY-1	P1	$U_{p_1} = 0.32; U_{p_2} = 0; U_{p_3} = 0$
SENSOR-3	P1	$U_{p_1} = 0.48; U_{p_2} = 0; U_{p_3} = 0$
SENSOR-1	P2	$U_{p_1} = 0.68; U_{p_2} = 0; U_{p_3} = 0$
SENSOR-2	P2	$U_{p_1} = 0.68; U_{p_2} = 0.4; U_{p_3} = 0$
DISPLAY-2	P3	$U_{p_1} = 0.68; U_{p_2} = 0.4; U_{p_3} = 0.32$

Table 3.4 shows that after the maps of task2 to processor p_1 the agent chooses to place task4 in processor p_1 instead of placing task 2 with the biggest utility U_{T_2} like in Table 3 this can be explained by the fact that the agent searches for the maximization of its reward and if it places task2 it will place task 1 after than it will be obliged to use a new processor since (U_{p_1} is inferior to all the other remaining tasks U_{T_j}) and thus following equation 6 it will receive a penalty (-1).

Following this idea the agent places task4 on p_1 and in consequently receives the maximum reward that it can collect in this step. Note that the new deployment model is different from the old one and needs three processors instead of two to correctly run.

We can also notice that the executions time of filling of Q-table values and the searching for the optimal solution is equal in both example, So two different cases with the same number of tasks and of processor but with different tasks features has the same execution time with or without the generate state phase. We can conclude that the execution time depends only on the number of tasks and processors.

3.3.2 Analysing The Q-table

In this section we are going to analyse the Q-table we got from our last example(after filling it) 3.3 and explain how we get the optimal solution using it.

Every column represents an action pair(T_n,P_n) with T_n the task number and P_n the processor number.For example (2,1) refers to placing task2 T_2, in processor1 P_1.

In this use case we have 5 tasks and 3 processors making the total number of actions equal to 15: (1,1) (1,2) (1,3) (2,1) (2,2) (2,3) (3,1) (3,2) (3,3) (4,1) (4,2) (4,3) (5,1) (5,2) (5,3).

Every Processor is represented using "[ ]" and every numbers represents a task. So in this example [1][3] we have T_1 placed in P_1 and T_3 placed in P_2.

The agent starts from the initial state [ ][ ][ ] where all the processors are empty and chooses the action with the Biggest Q-value.

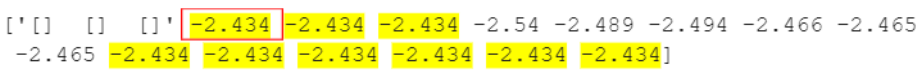

Figure 3.1: Initial State Q-values.

All the yellow cells are possible actions.

In our solution the agent always chooses the first biggest value in the table. Therefor it's going to choose action (1,1), so then it passes to state [1][ ][ ] and chooses the first action with the biggest Q-value. It continues doing that till all the tasks are placed.

These are all the states the agent passed through till getting to the optimal one :

```
['[]  []  []' -2.434 -2.434 -2.434 -2.54 -2.489 -2.494 -2.466 -2.465
 -2.465 -2.434 -2.434 -2.434 -2.434 -2.434 -2.434]

['[1]  []  []' -1000.0 -1000.0 -1000.0 -2.33 -2.63 -2.632 -2.327 -2.59
 -2.568 -2.167 -2.585 -2.587 -2.167 -2.585 -2.58]

['[1 3]  []  []' -1000.0 -1000.0 -1000.0 -9000.0 -2.492 -2.504 -1000.0
 -1000.0 -1000.0 -2.484 -2.725 -2.733 -2.484 -2.495 -2.507]

['[1 3 4]  []  []' -1000.0 -1000.0 -1000.0 -10001.12 -3.08 -3.08
-1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -10001.12 -3.08 -3.08]

['[1 3 4]  []  []' -1000.0 -1000.0 -1000.0 -10001.12 -3.08 -3.08
-1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -10001.12 -3.08 -3.08]

['[1 3 4]  [2]  []' -1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0
 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -10000.0 -10000.0
-2.1]

['[1 3 4]  [2]  [5]' -1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0
-1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0 -1000.0
-1000.0]
```

Figure 3.2: How The Agent Iterates the Q table.

3.3.3 Experimentation and evaluation

In this section we are going to assess the effectiveness of our approach and to demonstrate the great benefits collected from the use of Q-learning especially for features refactoring.

nt \ np	2	3	4	5	6
1	3	4	5	6	7
2	9	16	25	36	49
3	27	64	125	216	343
4	81	256	625	1296	2401
5	243	1024	3125	7.776	
6	729	4096	15625		
7	2187	16384			
8	6561				

Figure 3.3: Number of States

Table corresponding to figure 3.3 presents the number of states for a given (n,m) pair where n is the number of tasks ans m is the number of processors used.

Black cells are unreachable cases where our system crashes due to time and hardware limitation.

Using these result we can determine that the number of states for a certain pair(n,m) is equal to $(m+1)^n$.

The maximum possible number of states that our solution can generate for a given (n,m) pair is 16384 with n=7 and m=3.

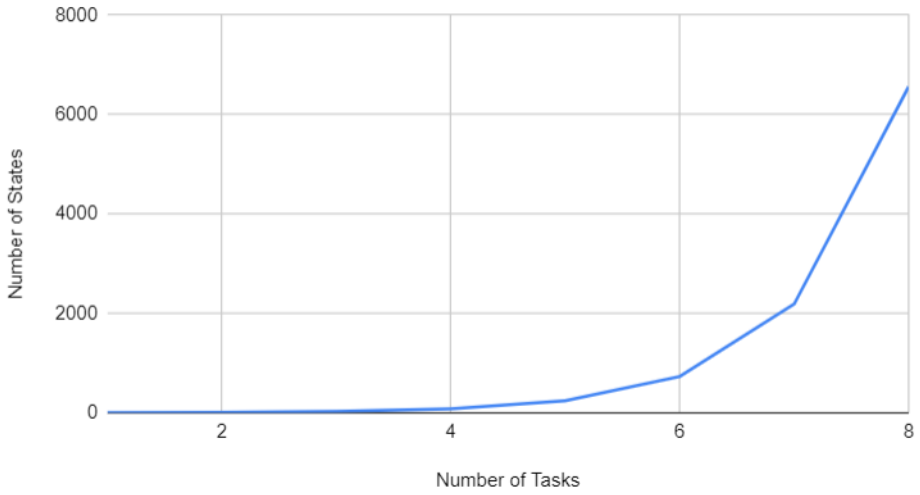

Figure 3.4: Number of State in a Case with 2 Processors.

Curve relative to figure 3.4 shows how the number of states varies with the variation of the tasks number for a case with 2 processors.

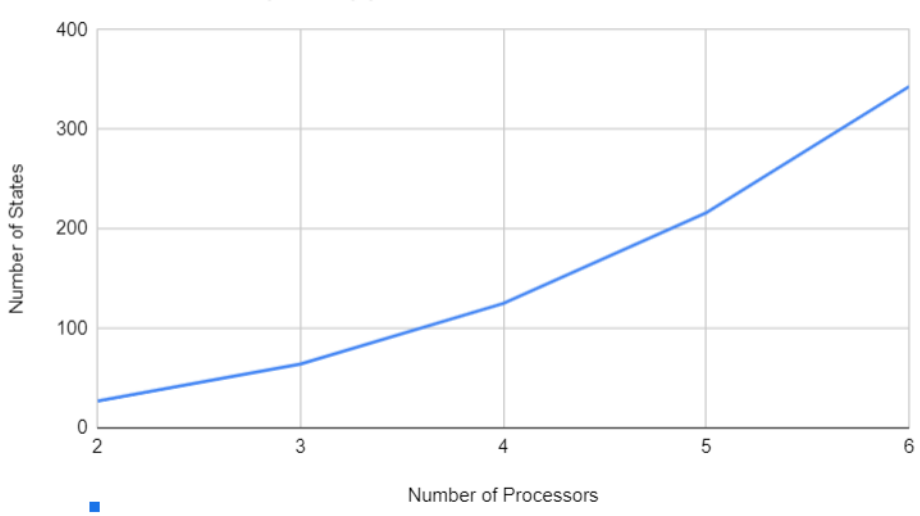

Figure 3.5: Number of State in a Case With 3 Tasks.

Curve relative to figure 3.5 shows how the number of states grows when the number of processors increases for a case with a constant number of tasks equal to 3.

We can conclude that the states number grows enormously with the number of tasks and processors.

These results can also explain the increase of the execution time taken to generate the states as shown by this figure:

n \ m	2	3	4	5	6
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0.015	0.031	0.046
4	0.015	0.062	0.25	0.89	2.84
5	0.093	1.58	14.39	82.65	
6	1.90	85.21	1,494.32		
7	72.54	7200			
8	3074.79				

Figure 3.6: The Execution Time Taken to Generate The States of a (m,n).

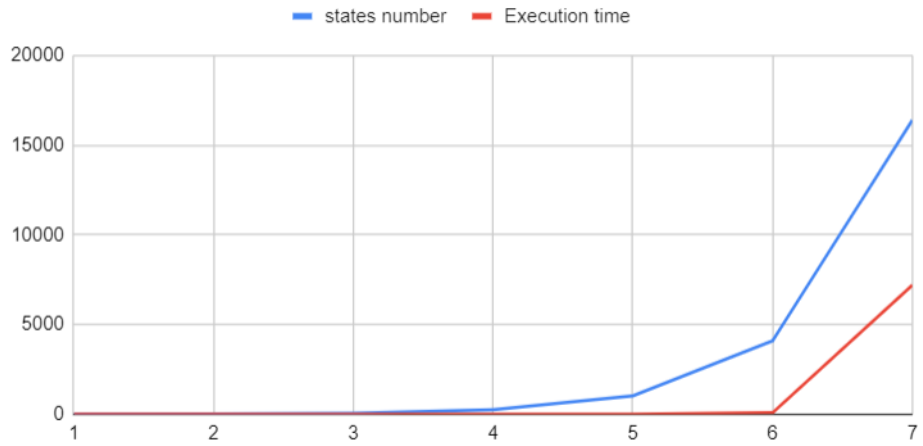

Figure 3.7: The Execution Time Taken to Generate The States VS The States Number In a Case With 3 processors.

The curve corresponding to figure 3.7 shows how the time taken to generate the states increases with the increase of the states number.

As a final conclusion the number of states increases with the increase of the tasks and processors numbers and with the increase of the states number the time taken to generate these states grows exponentially.

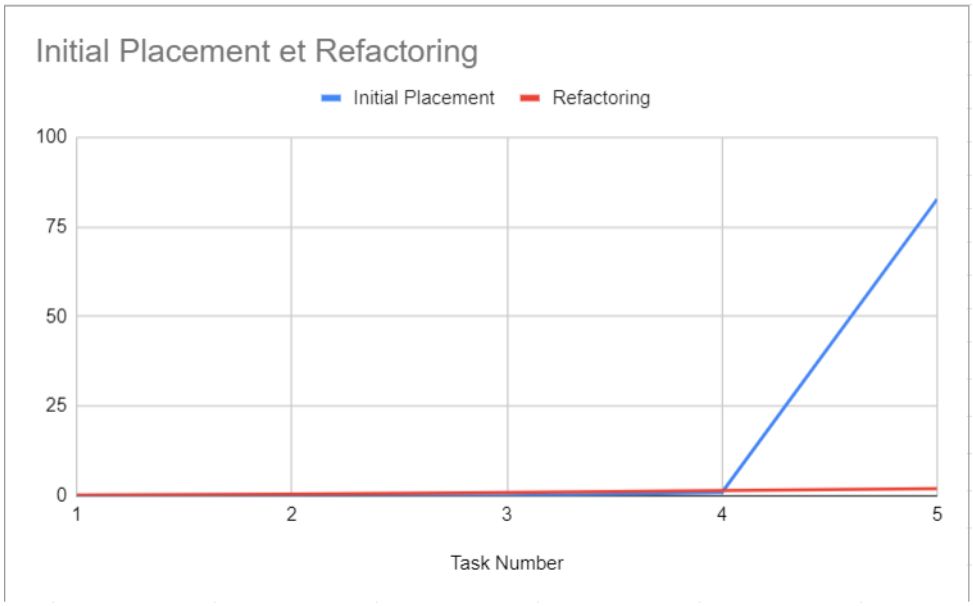

Figure 3.8: Execution Time for initial placement and Refactoring in a Case with 5 processors.

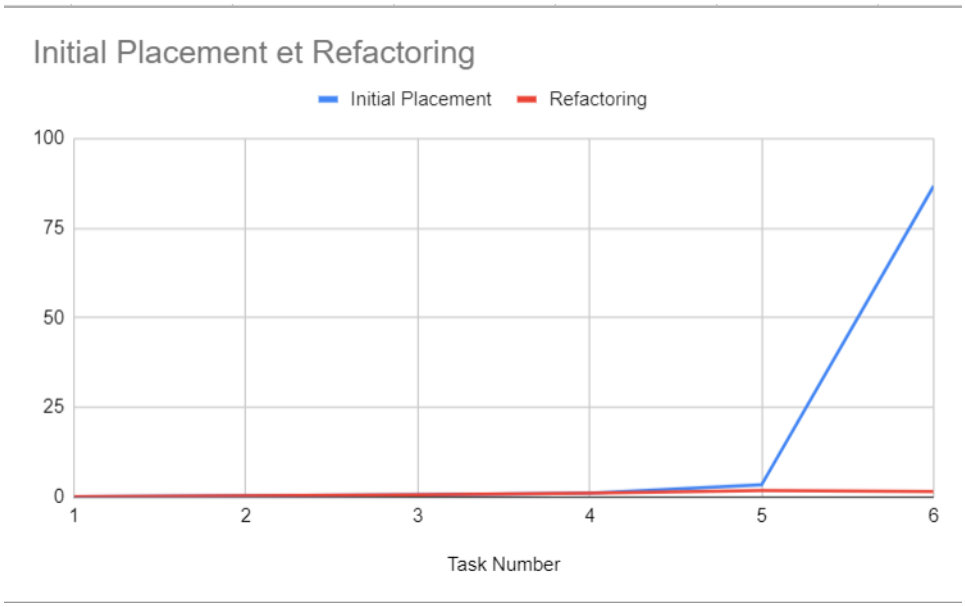

Figure 3.9: Execution Time for initial placement and Refactoring in a Case with 3 processors.

Curves relative to figures 3.8 shows results of experiments done on a randomly set of tasks where we fix in each step the number of processors and we progressively increment the tasks number.

We first remark that the peak, witch corresponds to the reaching of an enough great number of states, differs from one curve to another and it is obtained at different number of used tasks. This is explained by the fact that peaks depends on both the number of tasks and the one of processors thus peak for curves of figures 3.8 with a relatively superior processors number is reached before the curves of figures 3.9 with inferior ones. An important result directly concluded from figure 3.8 is

that the majority part of execution time is dedicated to the phase of states generation.

n \ m	2	3	4	5	6
1	0	0	0	0	0
2	0.1	0.1	0.1	0.1	0.1
3	0.1	0.2	0.6	1.4	2.8
4	0.4	2.7	7.5	22.4	53.2
5	3.8	35.6	183.9	682.7	
6	43.9	637.1	4362.9		
7	347.7				
8	9,779.6				

Figure 3.10: Memory Space Used To Generate the Sates.

The table relative to figure 3.10 shows that generating states is very memory demanding. Thus our approach is very beneficial in terms of time, RAM-memory and design for application with great rate of features factoring where states generation is inherited from the first application execution.

3.4 Conclusion

In this final chapter we have presented and analysed our solution using Q-learning algorithm and we have showed the great benefit to features refactoring , however this solution has some limitations. In fact the generation of state became unfeasible with the growth of the number of tasks and processors.

To deal with this problem we propose the use of The Deep RL that can be an immediate extension of our work.

General Conclusion

Our project is elaborated within The Higher Institute of Computer Science. It aims to create a model for finding the optimal solution to a placement problem in a distributed real-time system.

In this report, we have presented reinforcement Learning and we have decided to use Q-learning to create our model solution. Our system works perfectly, however, it has a maximum number of states that can generate due to time and hardware limitations add to that even if our model could generate all the states Q-learning becomes nearly impossible to converge due to the same limitations.

The best solution is to turn to deep reinforcement learning where we use a neural network that acts exactly like a Q-table. both Q-table and neural network are output a Q-Value with the state as input. Q-Learning requires enumerating all states in Q-table to converge, it requires very large memory and a very long training time in order to enumerate and explore all possible states. In fact in some complicated problems, it is possible that the q-table doesn't converge, simply because of so many states. In comparison, Neural networks can generalize the states and find a function that maps a state to its Q-value. It no longer needs to enumerate, it instead learns information implied in states. Even if the agent has never explored a certain state in training, as long as the neural network has been trained to learn the relationship with other similar states, it can still generalize and output the Q-value. That's why it is a lot easier to converge.

Bibliography

[1] *Isi presentation"*, [Accessed 16-May-2022]. [Online]. Available: `http://www.isi.rnu.tn/institut/presentation/`.

[2] *Utm presentation*, [Accessed 16-May-2022]. [Online]. Available: `http://www.utm.rnu.tn/utm/fr/universite--presentation`.

[3] *Utm ranking"*, [Accessed 16-May-2022]. [Online]. Available: `https://fr.wikipedia.org/wiki/UniversitC3A9_de_Tunis_-_El_Manar`.

[4] Anonymus, *Real-time systems: the international journal of time-critical computing systems*. Springer, 1989, [Accessed 13-March-2022].

[5] *Difference between soft, hard and firm real-time system*, [Accessed 7-March-2022]. [Online]. Available: `https://www.shresthanischal.com.np/2021/02/difference-between-soft-hard-and-firm.html`.

[6] *Notation for modeling real time task scheduling*, [Accessed 10-March-2022]. [Online]. Available: `http://www.jot.fm/issues/issue_2006_05/article2/`.

[7] *Difference between sporadic and aperiodic real-time tasks*, [Accessed 9-March-2022]. [Online]. Available: `https://www.geeksforgeeks.org/difference-between-sporadic-and-aperiodic-real-time-tasks/`.

[8] J. Korst, E. Aarts, and J. K. Lenstra, "Scheduling periodic tasks", *INFORMS journal on Computing*, vol. 8, no. 4, pp. 428–435, 1996, [Accessed 15-March-2022].

[9] Y. Solihin, *Fundamentals of parallel multicore architecture*. CRC Press, 2015, [Accessed 19-March-2022].

[10] B. Burns, *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. " O'Reilly Media, Inc.", 2018, [Accessed 20-March-2022].

[11] *What is refactoring and why is it important?*, [Accessed 4-May-2022]. [Online]. Available: `https://www.cuelogic.com/blog/what-is-refactoring-and-why-is-it-important`.

[12] O. Goldreich, *P, NP, and NP-Completeness: The basics of computational complexity*. Cambridge University Press, 2010, [Accessed 8-May-2022].

[13] R. Devillers and J. Goossens, "Liu and layland's schedulability test revisited", *Information Processing Letters*, vol. 73, no. 5-6, pp. 157–161, 2000, [Accessed 20-March-2022].

[14] N. R. C. E. John McCarthy Marvin L. Minsky, *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*. August 31, 1955, [Accessed 6-March-2022].

[15] S. Thrun and M. L. Littman, "Reinforcement learning: an introduction", *AI Magazine*, vol. 21, no. 1, pp. 103–103, 2000, [Accessed 1-April-2022].

[16] J. Lee and R. S. Sutton, "Policy iterations for reinforcement learning problems in continuous time and space—fundamental theory and methods", *Automatica*, vol. 126, p. 109 421, 2021, [Accessed 27-March-2022].

[17] P. Winder, *Reinforcement learning*. O'Reilly Media, 2020, [Accessed 9-April-2022].

ملخّص

يهدف هذا المشروع إلى تنفيذ نموذج التعلم المعزز لوضع المهام في الوقت الفعلي

كلمات مفاتيح : التعلم المعزز ، أنظمة الوقت الحقيقي ، الأنظمة الموزعة ، ص- التعلم

Résumé

Ce projet vise à mettre en œuvre un modèle d'apprentissage par renforcement pour le placemnt des tâches en temps réel.

Mots clés : Apprentissage par renforcement, Systèmes temps-réel, Systèmes distribués, Q-learning.

Abstract

This project aims to implement a reinforcement learning model for real-time tasks placement.

Keywords : Reinforcement learning, Real-Time systems, Distributed Systems, Q-learning.

2 نهج أبو الريحان الباروني 2080 أريانة الهاتف : 71 706 164 الفاكس : 71 706 698 البريد الالكتروني : isi@isim.rnu.tn

2, Abou Raihane Bayrouni 2080 l'Ariana **Tél** : 71 706 164 **Fax** : 71 706 698 **Email** : isi@isim.rnu.tn