Model-based reinforcement learning for dynamic resource allocation in cloud environments

Thomas Tournaire

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAS004

Thèse de doctorat

Model-Based Reinforcement Learning for Dynamic Resource Allocation in Cloud Environments

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Telecom Sud Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Mathématiques et Informatique

Thèse présentée et soutenue à Paris, le 02 Mai 2022, par

Thomas Tournaire

Composition du Jury :

André-Luc Beylot Professeur, INP ENSEEIHT, Toulouse	Président
Nicolas Gast CR Inria Grenoble LIG, HDR	Rapporteur
Sara Alouf CR Inria Sophia-Antipolis NEO, HDR	Rapporteure
Tijani Chahed Professeur, Telecom Sud Paris (SAMOVAR)	Examinateur
Bruno Tuffin DR Inria Rennes DIONYSOS	Examinateur
Sandrine Vaton Professeure, IMT Atlantique	Examinatrice
Hind Castel-Taleb Professeure, Telecom Sud Paris (SAMOVAR)	Directrice de thèse
Emmanuel Hyon MCF, UPL Paris Nanterre)	Co-directeur de thèse
Armen Aghasaryan Ingénieur de recherche et chef d'équipe, Nokia Bell Labs France	Invité

ABSTRACT

The emergence of new technologies (Internet of Things, smart cities, autonomous vehicles, health, industrial automation, ...) requires efficient resource allocation to satisfy the demand. These new offers are compatible with new 5G network infrastructure since it can provide low latency and reliability. However, these new needs require high computational power to fulfill the demand, implying more energy consumption in particular in cloud infrastructures and more particularly in data centers. Therefore, it is critical to find new solutions that can satisfy these needs still reducing the power usage of resources in cloud environments. In this thesis we propose and compare new AI solutions (Reinforcement Learning) to orchestrate virtual resources in virtual network environments such that performances are guaranteed and operational costs are minimised. We consider queuing systems as a model for clouds IaaS infrastructures and bring learning methodologies to efficiently allocate the right number of resources for the users. Our objective is to minimise a cost function considering performance costs and operational costs. We go through different types of reinforcement learning algorithms (from model-free to relational model-based) to learn the best policy. Reinforcement learning is concerned with how a software agent ought to take actions in an environment to maximise some cumulative reward. We first develop queuing model of a cloud system with one physical node hosting several virtual resources. On this first part we assume the agent perfectly knows the model (dynamics of the environment and the cost function), giving him the opportunity to perform dynamic programming methods for optimal policy computation. Since the model is known in this part, we also concentrate on the properties of the optimal policies, which are threshold-based and hysteresis-based rules. This allows us to integrate the structural property of the policies into MDP algorithms. After providing a concrete cloud model with exponential arrivals with real intensities and energy data for cloud provider, we compare in this first approach efficiency and time computation of MDP algorithms against heuristics built on top of the queuing Markov Chain stationary distributions. In a second part we consider that the agent does not have access to the model of the environment and concentrate our work with reinforcement learning techniques, especially model-based reinforcement learning. We first develop model-based reinforcement learning methods where the agent can re-use its experience replay to update its value function. We also consider MDP online techniques where the autonomous agent approximates environment model to perform dynamic programming. This part is evaluated in a larger network environment with two physical nodes in tandem and we assess convergence time and accuracy of different reinforcement learning methods, mainly model-based techniques versus the state-of-the-art model-free methods (e.g. Q-Learning). The last part focuses on model-based reinforcement learning techniques with relational structure between environment variables. As these tandem networks have structural properties due to their infrastructure shape, we investigate factored and causal approaches built-in reinforcement learning methods to integrate this information. We provide the autonomous agent with a relational knowledge of the environment where it can understand how variables are related to each other. The main goal is to accelerate convergence by : first having a more compact representation with factorisation where we devise a factored MDP online algorithm that

we evaluate and compare with model-free and model-based reinforcement learning algorithms; second integrating causal and counterfactual reasoning that can tackle environments with partial observations and unobserved confounders.

Keywords. Reinforcement learning, Markov Decision Process, Queuing systems, Factored reinforcement learning, Causal reinforcement learning, Auto-scaling policies, Cloud

RÉSUMÉ EN FRANÇAIS

L'émergence de nouvelles technologies nécessite une allocation efficace des ressources pour satisfaire la demande. Cependant, ces nouveaux besoins nécessitent une puissance de calcul élevée impliquant une plus grande consommation d'énergie notamment dans les infrastructures cloud et data centers. Il est donc essentiel de trouver de nouvelles solutions qui peuvent satisfaire ces besoins tout en réduisant la consommation d'énergie des ressources. Dans cette thèse, nous proposons et comparons de nouvelles solutions d'IA (apprentissage par renforcement RL) pour orchestrer les ressources virtuelles dans les environnements de réseaux virtuels de manière à garantir les performances et minimiser les coûts opérationnels. Nous considérons les systèmes de file d'attente comme un modèle pour les infrastructures cloud IaaS et apportons des méthodes d'apprentissage pour allouer efficacement le bon nombre de ressources. Notre objectif est de minimiser une fonction de coût en tenant compte des coûts de performance et opérationnels. Nous utilisons différents types d'algorithmes de RL (du « sans-modèle » au modèle relationnel) pour apprendre la meilleure politique. L'apprentissage par renforcement s'intéresse à la manière dont un agent doit agir dans un environnement pour maximiser une récompense cumulative. Nous développons d'abord un modèle de files d'attente d'un système cloud avec un nœud physique hébergeant plusieurs ressources virtuelles. Dans cette première partie, nous supposons que l'agent connaît le modèle (dynamiques de l'environnement et coût), ce qui lui donne la possibilité d'utiliser des méthodes de programmation dynamique pour le calcul de la politique optimale. Puisque le modèle est connu dans cette partie, nous nous concentrons également sur les propriétés des politiques optimales, qui sont des règles basées sur les seuils et l'hystérésis. Cela nous permet d'intégrer la propriété structurelle des politiques dans les algorithmes MDP. Après avoir fourni un modèle de cloud concret avec des arrivées exponentielles avec des intensités réelles et des données d'énergie pour le fournisseur de cloud, nous comparons dans cette première approche l'efficacité et le temps de calcul des algorithmes MDP par rapport aux heuristiques construites sur les distributions stationnaires de la chaîne de Markov des files d'attente. Dans une deuxième partie, nous considérons que l'agent n'a pas accès au modèle de l'environnement et nous concentrons notre travail sur les techniques de RL. Nous évaluons d'abord des méthodes basées sur un modèle où l'agent peut réutiliser son expérience pour mettre à jour sa fonction de valeur. Nous considérons également des techniques de MDP en ligne où l'agent autonome approxime le modèle pour effectuer une programmation dynamique. Cette partie est évaluée dans un environnement plus large avec deux nœuds physiques en tandem et nous évaluons le temps de convergence et la précision des différentes méthodes, principalement les techniques basées sur un modèle par rapport aux méthodes sans modèle de l'état de l'art. La dernière partie se concentre sur les techniques de RL basées sur des modèles avec une structure relationnelle entre les variables d'état. Comme ces réseaux en tandem ont des propriétés structurelles dues à la forme de l'infrastructure, nous intégrons les approches factorisées et causales aux méthodes de RL pour inclure cette connaissance. Nous fournissons à l'agent une connaissance relationnelle de l'environnement qui lui permet de comprendre comment les variables sont reliées. L'objectif principal est d'accélérer la convergence : d'abord avec une représentation plus compacte avec

la factorisation où nous concevons un algorithme en ligne de MDP factorisé que nous comparons avec des algorithmes de RL sans modèle et basés sur un modèle ; ensuite en intégrant le raisonnement causal et contrefactuel qui peut traiter les environnements avec des observations partielles et des facteurs de confusion non observés.

Mots-clés. Apprentissage par renforcement, Processus de Décisions Markoviens, Files d'attente, Apprentissage par renforcement factorisé, Apprentissage par renforcement causal, Politiques d'auto-scaling, Cloud

ACKNOWLEDGEMENT

It will be very difficult for me to thank everyone because it is thanks to the help of many people that I was able to complete this thesis.

First of all, I would like to thank all the members of the Jury (Mr. Andre-Luc Beylot, Mrs. Sandrine Vaton, Mr. Tijani Chahed, Mr. Bruno Tuffin) who participated in the thesis defense, as well as the two reviewers Mrs. Sara Alouf and Mr. Nicolas Gast for their detailed reports and their questioning which opened new perspectives.

I would also like to thank my thesis supervisors, Mrs. Hind Castel-Taleb and Mr. Emmanuel Hyon, for all their help, their advice and for all that I could learn at their side. I am delighted to have worked with them because, in addition to their scientific support, they were always there to support and advise me during the development of this thesis.

I also thank my team from Nokia Bell Labs (Armen Aghasaryan, Luka Jakovjlevic, Yue Jin, Makram Bouzid, Dimitre Kostadinov, Antonio Massaro, Liubov Tupikina) with whom I could learn a lot on many different subjects.

I would like to thank the following people for giving me the opportunity to work on several Nokia use-cases : Fahad Syed Muhammad, Veronique Capdevielle, Afef Feki, and also François Durand and Lorenzo Maggi for their precious advices in these projects.

I wish to extend my special thanks to Lorenzo Maggi for his kindness, his advices and all the elements i have learned while working with him.

FInally, I must express my very profound gratitude to my parents Lydie and Bruno, my brother Paul and my Family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Last, I thank my wife Marie-Anne who has always been present during these three years of thesis, for her moral support, her assistance for the repetition of presentations, and also her sharing of motivation. This accomplishment would not have been possible without them.

Thank you.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

NOMENCLATURE

Queuing systems

λ	Arrival rate in queuing systems
μ	Service rate in queuing systems
Π	Stationary distribution of the macro chain
π	Markov Chain stationary distribution
π_k	Stationary distribution of a micro chain of level k
B_i	Maximum number of requests in the buffer of physical node i
C_A	Cost of activating one VM
C_D	Cost of deactivating one VM
C_H	Cost of holding for requests in system within 1 time unit
C_R	Cost of rejecting one request due to full queue
C_S	Cost of one working VM within one time unit
F_k	Number of requests such that we activate a VM
K	Total number of virtual resources (VMs) in physical node i
k_i	Number of activated virtual resources in physical node i
m_i	Number of jobs in physical node i
q	Control policy
R_k	Number of requests such that we deactivate a VM

Reinforcement Learning

$\bar{\mathcal{P}}$	Uniformised probability distribution
$\bar{\mathcal{R}}$	Uniformised reward function
$\mathcal{A}$	Action space
$\mathcal{P}$	Probability transition

$\mathcal{Q}$	Action-value function
$\mathcal{R}$	Reward function
$\mathcal{S}$	State space
$\mathcal{V}$	State-value function
$\tilde{\mathcal{P}}$	Approximated probability distribution
$\tilde{\mathcal{R}}$	Approximated reward function
q	Policy

PART I

DYNAMIC CLOUD MANAGEMENT

CHAPTER 1

INTRODUCTION

1.1 Motivation

The emergence of new technologies (Internet of Things IoT, smart cities, autonomous vehicles, health, industrial automation, augmented reality, ...) requires a growing need for network and cloud to satisfy low latency and reliability. These emerging technologies will increase the resource requirements by requiring high computational power and therefore the energy consumption of the networks. This phenomenon is receiving a lot of attention from the world's governments regarding the need to reduce the energy consumption of the cloud. Indeed, the theme of energy-efficient cloud computing is now a priority on the European Union (EU) political agenda. In 2018, the energy consumption of data centers in the EU was 76.8 TWh and is expected to rise to 98,52 TWh by 2030, a 28% increase. In addition, the Information and Communication Technology (ICT) sector is a major energy consumer, using about 4% of the world's electricity today and expected to account for 10-20% of global electricity consumption by 2030. Also, the global digital ecosystem is responsible for 2% to 4% of the world's greenhouse gas emissions, up to twice as much as air travel [1].

There is therefore an urgent need to find intelligent resource allocation policies that can reduce energy consumption and satisfy modern and future needs. Queuing theory is a branch of mathematics that studies how queues form, how they work and why they don't work. It looks at every component of waiting in line, including the arrival process, the service process, the number of servers, the number of seats in the system, and the number of customers - which can be people, data packets, cars, etc. Queuing theory is often used to represent, analyse and optimise Cloud systems, by representing such infrastructures with multiple nodes (physical servers, virtual resources, etc.) in network. Cloud infrastructures are physical nodes hosting virtual resources, connected in a network. Figure 1.1 is a very small representation of such systems where User's Equipments (UEs) send requests to be processed by the Cloud and asking for a response. The queuing models allow researchers to evaluate performance metrics such as number of requests (or customers), delays, losses, CPU utilisation rate, energy consumption, etc.

Abstracting cloud infrastructures and networks with queuing models allows us to devise and assess new intelligent algorithms by evaluating their performance. The main objective of this thesis is to provide and assess algorithms based on artificial intelligence (AI) that can learn and update efficient policies continuously for resource allocation. These learned policies (or solutions) are rules that tell the agent what decision to make for an observation of the cloud system. Activation and deactivation of resources (virtual or physical) is an indispensable part of energy management and our goal is therefore to consider such activation/deactivation rules in Cloud sys-

1. Guillaume Pitron, L'enfer numérique - voyage au bout d'un like, Les Liens Qui Liberent, 2021

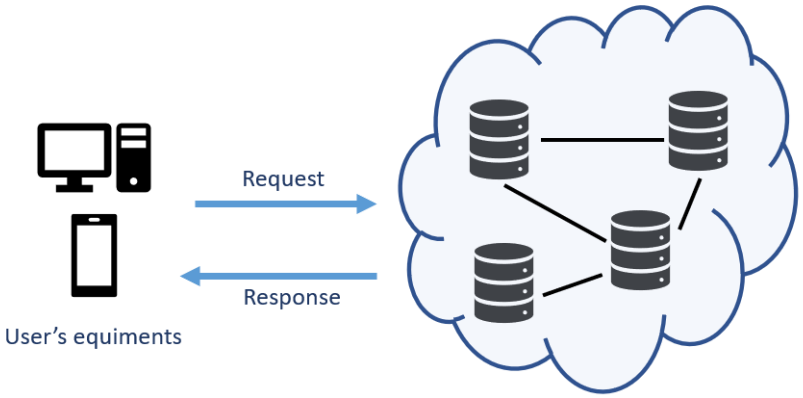


FIGURE 1.1 – Principal type of networks considered in the thesis

tems. In a nutshell, we provide and evaluate resource allocation solutions that can dynamically turn On or Off Cloud resources given the load (number of requests) of the system in order to optimise energy consumption and clients performances.

We first provide a fairly comprehensive review of energy management in Cloud environments by describing Cloud systems, the energy management and existing solutions to devise optimal policies that can reduce energy consumption. Analysis of the different existing techniques led us to explore the Reinforcement Learning techniques that can handle scenarios where the learning agent does not have all the knowledge about the system, which is often the case in practice.

Therefore, in this document, the principal framework we consider is Markov Decision Process (MDP), which is the underlying mathematical formalism for Reinforcement Learning (RL). This document describes the different RL techniques applied to queuing systems to learn autonomously new efficient resource allocation policies in complex Cloud environments. We talk about complex Cloud systems because we will tackle large state spaces by considering environments with many requests and many resources. In this context, minimising a cost function that takes into account energy and performance for clients is a NP-hard problem. Indeed, finding a rule or an optimal decision for all possible states of the system is highly difficult and requires the study of efficient algorithms.

This is why this document covers a large comparison of RL techniques by considering different assumptions that can be find in real cloud scenarios. These assumptions deal with the autonomous agent's knowledge of the environment. Indeed, our first part is devoted to cases where the agent has a full knowledge of the environment and can apply classical optimisation techniques such as heuristics coupled with Markov Chain (MC) analysis or MDP algorithms. In this context, we focus ourselves on structural properties of the policy, such as hysteresis and threshold policies implemented by leading cloud providers such as Amazon AwS EC2 or Microsoft Azure. Moreover we describe a real cost function integrating energy and performance data to assess the algorithms.

Next, we restrict ourselves in scenarios where the autonomous agent (or human experts) does not know perfectly the environment and require RL techniques to overcome this issue. We investigate different model-based reinforcement learning methods to compare with model-free RL techniques such as Q-Learning or Deep-Q-Network, which are widely used in practice. Our goal is to give the ability to the agent to learn a model of the environment to perform planning and overcome convergence issues in standard model-free RL. Finally, we consider scenarios where the agent understands how environment variables are related and devise new structural RL algorithms based on Factored RL (FRL) and Causal RL (CRL) frameworks. Overall, the evaluation of the RL algorithms is done by looking at their convergence speed and their accuracy on different state space scale, which are important metrics in real applications. Indeed, it is important for Cloud providers or clients to compute the best policy that can reduce energy consumption while maintaining good performances but also to do the calcu-

lation rapidly to cope with fast changing environments. Thus the comparison of RL methods is done based on these two elements : accuracy and convergence speed.

For short, this thesis includes two major components : searching for auto-scaling solutions in cloud network systems and building more efficient RL/AI algorithms. We have explored and searched among many techniques that allow to go beyond the classical state-of-the-art (SoTA) RL and they are described in this document. This research is divided in two main axes which are located in two parts : model-based RL and structural model-based RL.

1.2 Contributions of the thesis

The contributions of this thesis are as follows :

* Chapter II presents an overview of state of the art (SoTA) techniques for Cloud network modelling, Cloud auto-scaling, Cloud energy management and optimisation with machine learning techniques.
* Chapter III introduces the formalism of reinforcement learning and discuss many state of the art techniques. This review of the reinforcement learning literature was assisted by concrete studies in the network field, mainly with model-free RL techniques. These studies were conducted at Nokia Bell Labs and allowed me to work on two patents.

  * F. Syed Muhammad, R. Damodaran, T. Tournaire, A. Feki, L. Maggi, F. Durand, U. Chowta. *AI/ML based PDCP Split in Dual/Multi Connectivity Configurations : Collective Inference/Decision and Impact on 3gpp Interfaces*, Nokia, 2021.
  * T. Tournaire, F. Syed Muhammad, A. Feki, L. Maggi, F. Durand. *Multi-Agent Reinforcement Learning Sharing Common Neural Network Policy for PDCP Data Split*, Nokia, 2021.

The *Part II* of the thesis deals with *model-based* reinforcement learning domain for computing and learning auto-scaling policies in Cloud environments :

* Chapter IV is about auto-scaling threshold policies in Cloud systems when the agent knows the environment model. It studies and compares two mathematical approach to find optimal threshold policies in a Cloud environment, modelled with multi-server queuing systems : Heuristics based on the computation of stationary distribution of the Markov chain and Markov Decision Process. It compares many local search heuristics and proposes many improvement techniques : Initialisation of the heuristics, Aggregation technique for fast cost computation ; Meta-heuristic. Next, it studies meticulously the Markov Decision Process approach and designs new Dynamic Programming algorithms integrating the structure of the policy, namely hysteresis and thresholds properties. Finally it proposes a real cloud scenario which features real energy, financial data and requests arrivals. All techniques are compared numerically.

  This chapter has led to two publications [139, 141] :

  * Thomas Tournaire, Hind Castel-Taleb, Emmanuel Hyon. Generating optimal thresholds in a hysteresis queue : a cloud application. In *27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 19)*, 2019.
  * Thomas Tournaire, Hind Castel-Taleb, Emmanuel Hyon. Optimal control policies for resource allocation in the Cloud : comparison between Markov decision process and heuristic approaches. In *CoRR abs/2104.14879*, 2021.

The remaining chapters treats about scenarios where the agent has some partial information about environment dynamics and need to devise reinforcement learning techniques. Moreover, the following chapters consider a more complex Cloud environment by considering multi-tier networks modelled by queues in tandem.

* Chapter V aims to learn environment model for planning and deals with model-based reinforcement learning techniques. The aim is to compare model-based and model-free techniques for learning auto-scaling process in multi-tier Cloud architectures. It also investigates a partially observable environment to assess the robustness of model-based RL techniques. A precise comparison of different RL algorithms is described and numerical results are displayed in a discrete event simulator.

  The contributions of this chapter have been published in [142] :

  * Thomas Tournaire, Jeanne Barthélémy, Hind Castel-Taleb, Emmanuel Hyon. Reinforcement Learning with Model-Based Approaches for Dynamic Resource Allocation in a Tandem Queue. In *Performance Engineering and Stochastic Modeling. Springer International Publishing (ASMTA 21)*, 2021

The *Part III* is in the continuity of Chapter V except that it integrates a knowledge of the structure of the environment with two approaches : Factored and Causal.

* Chapter VI presents benefits having structural knowledge of the environment variables with factored representation. It proposes a factored reinforcement learning algorithm and a factored representation of the multi-tier network environment. Our algorithm is finally compared with SoTA RL techniques in a simulated environment and shows improvement when providing additional knowledge to the learning agent.

  The contributions of this chapter would be published in [140] :

  * Thomas Tournaire, Yue Jin, Armen Aghasaryan, Hind Castel-Taleb, Emmanuel Hyon. Factored Reinforcement Learning for Auto-scaling in Tandem Queues. In *Network and Service Management in the Era of Cloudification, Softwarization and Artificial Intelligence (NOMS 22)*, 2022

* Chapter VII goes further by considering *Causal Reinforcement Learning*. It proposes a unified overview of the whole Reinforcement Learning field and introduces causal (counterfactual) reasoning in RL solutions to improve current solutions.

  The contributions of this chapter will be soon published :

  * Thomas Tournaire, Armen Aghasaryan. Causal Reinforcement Learning for Auto-scaling in Tandem Queues, 2022

Finally, we summarise the thesis and discuss the ongoing and future investigations in Chapter VIII.

1.3 Structure of the thesis

To grasp the connections between the different chapters we provide how the document is structured presenting the chapters and their relations in Figure 1.2.

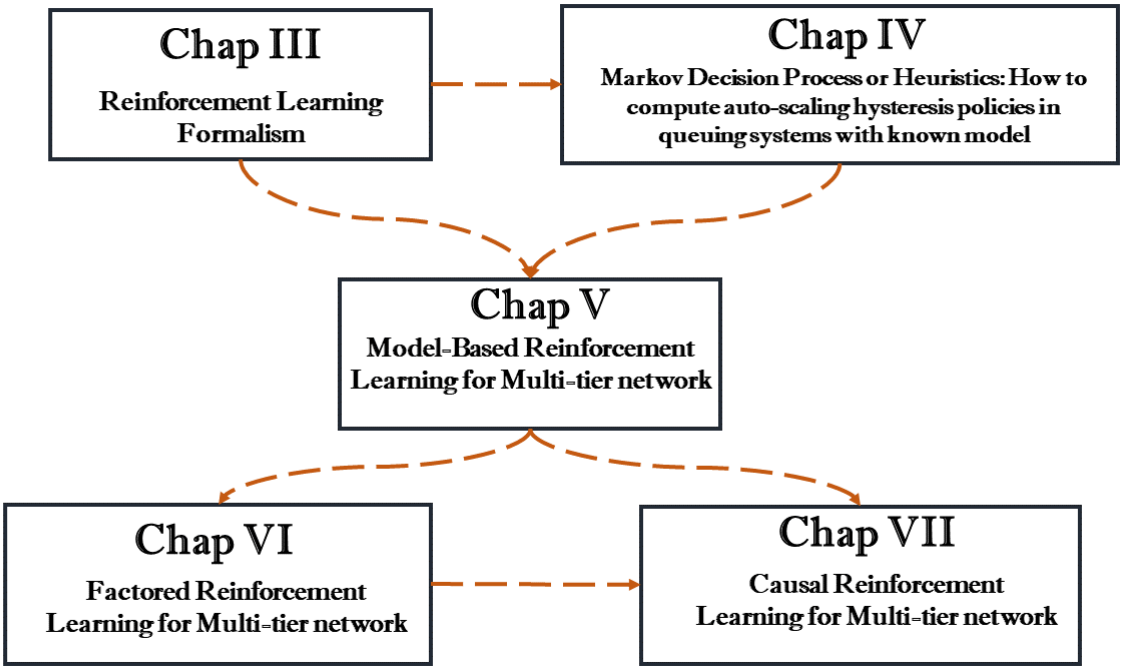


FIGURE 1.2 – Thesis flow chart

CHAPTER 2

ENERGY MANAGEMENT IN CLOUD ENVIRONMENTS

This chapter presents literature works about Cloud computing, energy management, queuing models, and optimisation techniques for resource allocation. It first presents what is Cloud computing, how it behaves, the existing infrastructures and models (e.g. datacenter) and the virtualisation framework (section 2.1). Next it gives an overview of energy management field by presenting the need, the solutions to measure and model, and dynamic resource allocation paradigm (section 2.2). Finally, it presents the existing auto-scaling techniques with thresholds-based rules, queuing models and control management and reinforcement learning techniques (section 2.3).

2.1 Cloud systems

2.1.1 Definition

Cloud Computing is a recent technology that allows access to data or infrastructure using an internet connection. This data is managed remotely by physical or virtual servers installed in a *datacenter*. This online storage space would go back, according to the newspaper [1] to the 1990's and the name *Cloud* would have been born in 2006 with the former CEO of Google, Eric Schmidt. Today, many companies have invested massively in Cloud Computing. Among the main companies of the sector are Amazon (AwS), Citrix, Google, IBM, Intel, Microsoft or OVHcloud. Users can then subscribe to different Cloud services, such as Amazon Drive, Google Drive, Microsoft OneDrive or for more professional applications with Microsoft Sharepoint for example and any applications to store companies data. In 2018 it was estimated that 3.6 billion people were accessing a huge range of cloud computing services.

2.1.2 Functioning of the Cloud

In order to operate, a Cloud needs to have several essential characteristics :

* The implementation of the systems is entirely automated and the customer can, according to the contract signed with the owner of the Cloud, have access thanks to the network to software or to his data when he wishes it and from anywhere in the world.

1. Technology Review

* Broadband network access : The datacenters that host these Clouds are generally connected directly to the Internet backbone (very powerful Internet network) to have excellent connectivity. The large providers then distribute different processing centers around the world to provide very fast access to systems for people around the world.
* Resource Reservoir : Most data centers contain tens of thousands of servers and storage facilities to allow for rapid scalability.
* Rapid scaling (elasticity) : Bringing a new server instance online is done in minutes, shutting down and restarting in seconds. These management techniques allow you to take full advantage of pay-per-use billing by adapting computing power to instantaneous traffic.
* The use of resources can be monitored and controlled, which makes it possible to optimize resource allocation for both the customer and the supplier.

In this thesis, we are mainly interested in the resizing of the Cloud and the measured service, i.e. the optimal policies of stop and restart of the servers (physical or virtual) so that the owner of the Cloud has a reduced financial cost as well as a reduction of its energy consumption.

2.1.3 Cloud architectures

2.1.3.1 Service models

There are different service models for the Cloud (depicted in Figure 2.1). Customers can use them in different ways and for different needs.

* Software as a Service (SaaS) : This service model is characterized by the use of a shared application that runs on a Cloud infrastructure. The application administrator does not manage or control the underlying infrastructure (networks, servers, applications, storage). A well-known example of SaaS is email software. These infrastructures provide the email service to billions of users.
* Platform as a Service (PaaS) : The user has the possibility of creating and deploying on a Cloud PaaS infrastructure his own applications by using the languages and the tools of the provider. He can also manage his data. However, he does not manage the underlying Cloud infrastructure (networks, servers, storage).
* Infrastructure as a Service (IaaS) : The user borrows computing and storage resources, network capacities and other essential resources (load balancing, firewall, etc.). The customer can deploy any type of software, including operating systems. The best known example today is Amazon Web Services which provides computing power (EC2), storage (S3, EBS) and online databases (SimpleDB).

2.1.3.2 Data centers : host of the Clouds

Data centers are infrastructures composed of a network of computers and storage spaces. This infrastructure can be used by companies to organise, process, store and warehouse large amounts of data. Cloud service providers use these data centers to host their equipment. Moreover, many companies host their infrastructure in the Cloud which avoids the management of the datacenter.

Clustering refers to the techniques of grouping together several independent computers called 'nodes' to allow global management and to go beyond the limits of one computer to :

* increase the availability ;
* facilitate the increase in load ;
* allow a distribution of the load ;

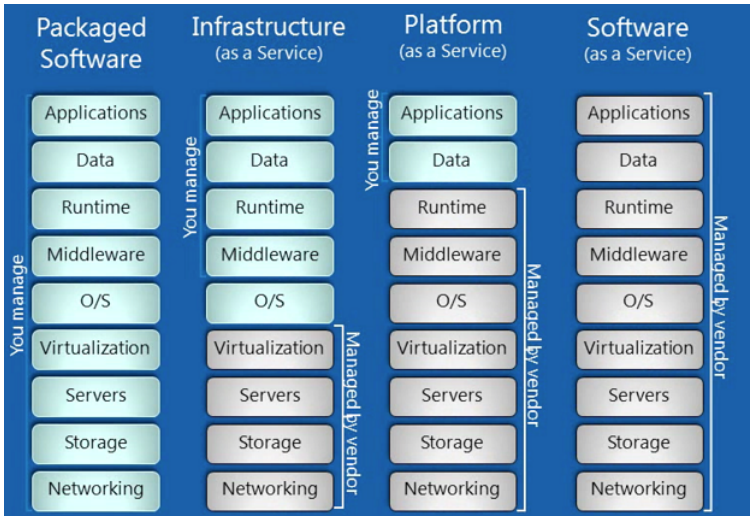


FIGURE 2.1 – Different Cloud Service Models

∗ facilitate the management of resources (processor, RAM, hard disks, network bandwidth).

The creation of clusters of servers is a process which is not very expensive, which consists in setting up several computers in network which will appear as a single computer to increase the performances (power of the processor, dimension of the storage space, quantity of RAM, etc.). This optimised use of resources allows the distribution of processing on the different machines.

One of the main advantages is that it is no longer necessary to invest in an expensive multi-processor server and that one can be satisfied with smaller devices which will then be connected to each other according to this 'cluster' principle, allowing a better adaptability according to the needs - from a performance as well as a financial point of view. This assembly of physical servers is very practical for the infrastructure. However, we can still improve the performance of the system by creating virtual servers.

2.1.3.3 Servers virtualisation

The **virtualisation of servers** [11, 64] allows to instantiate several virtual servers on a physical server. These virtual resources run on the same physical machine, while having the same properties as if they had each one a physical machine. The goal of this manipulation is to optimise the efficiency (computing power, scaling, costs) of a physical server while allowing the company to save money on physical infrastructure. The two principal virtual servers are docker containers and virtual machines (VMs). The main difference between a container and a virtual machine is that a container provides virtualisation at the operating system level, while a VM provides virtualisation at the hardware level (see Figure 2.2). In more detail : all containers share the host operating system while each virtual machine runs in its own operating system. In addition, the VM requires more memory space than a container and the startup time of a container is expressed in milliseconds whereas it takes several minutes for a VM. Note that nowadays most companies use these technologies to reduce costs and improve performance. Recent works [120] made performance comparison between these two virtualisation process.

This process allows a more efficient use of IT resources. Before server virtualisation, it was common to have over-used or under-used hardware in the same data center. With virtualisation, it is possible to move workloads between virtual machines with *migration*, allowing more flexibility in the workload management. Moreover, virtual servers can be easily adapted to the shifting demands of an organisation or private users. As virtual servers utilise the existing computational power of physical machines (vCPU), they can be scaled up or down, depen-

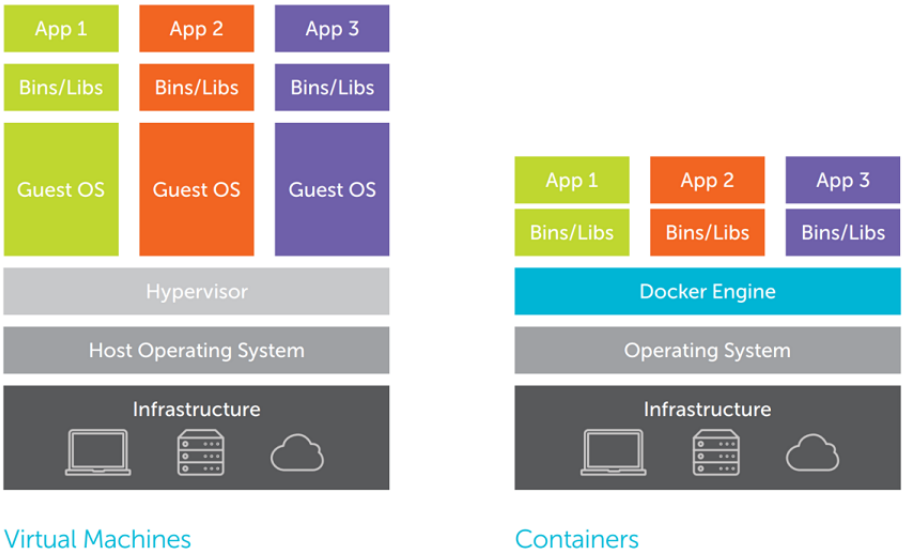


FIGURE 2.2 – Virtualisation process with VMs and containers

ding on the current needs. However, if you choose to split the power to share with other virtual servers, it can be distributed across multiple environments and help the physical appliance run more efficiently. This way, you can use the same processing power to run multiple workloads at the same time, without negatively impacting their performance. Another advantage of this technique is that virtual machines hardly ever fail, which greatly increases the reliability of the systems.

2.1.3.4 N-tier Software Architecture

N-tier architectures [2] are the main software application for client-server architectures. The most famous one is the 3-tier architecture. It is a software architecture where the application is decomposed into three logical tiers : the presentation layer or user interface, the application layer where data is processed and the data storage layer. This architecture is widely used in client-server applications such as a web applications. The main benefit of three-tier architecture is the local decomposition for each tier on their own infrastructure. Each tier treats a specific task and can be managed independently (scaling, updates, etc.). Currently multi-tier applications are subject to modernisation, using cloud-native technologies such as containers and can be instantiated in well-known cloud providers such as Amazon Web Scaling [15]. Such architectures can be modeled by queuing models especially networks of queues (see [143]).

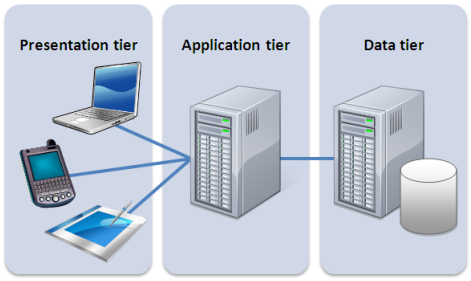


FIGURE 2.3 – Three tier architecture

2. IBM Cloud Learn Hub, https ://www.ibm.com/cloud/learn/three-tier-architecture, 2020

2.2 Energy management

2.2.1 The need for energy management

As we can see in Figure 2.4, representing the growth of data storage in the Amazon S3 cloud, cloud usage continues to grow each year, and exponentially so. This phenomenon can be observed in all companies with cloud services and is resulting in the construction of new data centers all over the world. This generates a massive use of data centers associated with the different clouds on the market, which implies a strong increase in their energy consumption (see Figure 2.4 [3]).

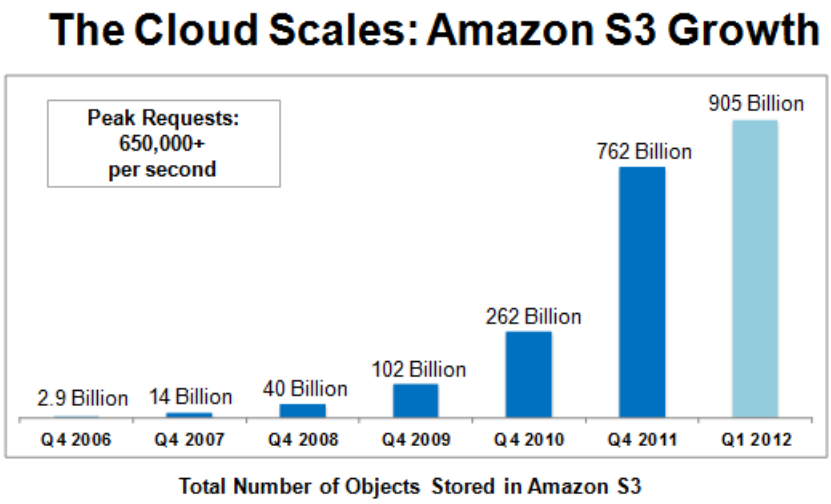


FIGURE 2.4 – Increase in data usage and Cloud

There is therefore an urgent need to fight against this over-consumption of energy, which is increasing year after year. Nowadays, Cloud computing requires more electric power than those of whole countries such as India or Germany [102], [4]. There are several ways to remedy this problem : improving air conditioning (free-cooling), improving physical servers or clustering, etc. But there also seems to be a role to play regarding the management of resources in a Cloud, in particular the activation/deactivation policies of physical or virtual servers.

Energy consumption increase is one of the many challenges facing large-scale computing. When the resource utilisation is too low, some of them can be turned off to save energy, without sacrificing performance. In a data center, the power consumption can be divided into static and dynamic parts. The static parts are the base costs of running the data center when being idle and the dynamic costs depend on the current usage. In [87], a power-aware model was defined to estimate the dynamic part of energy cost for a virtual machine (VM) of a given size, this model keeps the philosophy of the pay as you go model but based on energy consumption.

Performance and Energy trade-off However, managing energy is not without cost as it reduces the performance of the system. The Service Level Agreement (SLA) is a contract signed between the owner of the cloud and clients. It defines the performance qualities (*QoS*) allowed during the period of the contract, i.e. the service times of the servers, the repair times, a guarantee of data protection, etc. It is therefore a parameter to be taken into account when cloud providers or clients want to minimise the average global cost generated by the Cloud, because

3. Jeff Barr, Introduction to Amazon Web Services, slides
4. Le cloud, les data centers et l'énergie,Observatoire de l'industrie Electrique,2017, https ://observatoire-electricite.fr/usages-de-l-electricite/article/le-cloud-les-data-centers-et-l-energie

one cannot minimise the energy cost without losing performance. This is why it is necessary to balance the two metrics and find the best compromise. The SLA is a crucial component to consider for energy management yet it can be difficult to establish properly and several works ought to describe SLA in Cloud environments [126] [5]. Moreover, Siddesh et al. study in [128] dynamic resource allocation with risk analysis by meeting Service Level Agreements. Last, Labidi et al. [88] propose a semantically richer ontology-based model for SLA to improve its definition and its evaluation in cloud computing.

2.2.2 Measuring and modelling the energy

This section describes how the energy in cloud systems can be measured and modelled and also presents dynamic resource allocation.

2.2.2.1 Energy measurements

Being able to measure the energy consumption of cloud systems is crucial for the development of energy efficient policies. Indeed, such metrics are important to consider as inputs for scaling policy algorithms. However, this is not done by organisations in practice since only 13.4% of them are monitoring power consumption [152]. Yu et al. argue and propose in [152] what Key Performance Indicators (KPI) should be suitable for energy management. They define an Energy Management Protocol (EMP), which allows the recovery of information about energy consumption, as well as remote power management. Moreover, Kenga et al. [75] propose an experimental approach for measuring power consumption in IaaS cloud systems, using Intel's *Running Average Power Limit* framework. *Green Grid*, an international consortium [24] also presents two metrics to compute the amount of energy consumed by cloud systems : Power Usage Effectiveness (PUE) and Data Centre Infrastructure Efficiency (DCiE) which are formulas based on Total Facility Power and IT Equipment Power.

$$PUE = \frac{Total\ facility\ power}{IT\ equipment\ power}$$

The main consumers of energy on a server (physical or virtual) will be the CPU, the GPU, and the memory. Estimating how much each consumes will give you an estimate of how much power your server, or your application on a server is consuming. Kansal et al. [72] tackle the virtualisation scenario and provides a solution for VM power metering called *Joulemeter*. They build power models to infer power consumption from resource utilisation ad runtime. Next they demonstrate how current instrumentation for server hardware and hypervisors can be applied to build the power models on real platforms with very low error. The work [85] also provides a power model considering utilisation of specific isolated resources to measure consumption of single VM. They measure the per-VM utilisation of various resources. Finally, Orgerie et al. [87] were among the first to provide measures of the energy consumption of single VMs.

2.2.2.2 Energy models

The measurement of the energy of an entire cloud requires mathematical models or tools to represent the cloud as finely as possible to have very good approximation of the power usage. Indeed, measuring electricity with a watt meter does not necessarily identify which resource is currently consuming energy, specially in virtualised environments. Several authors proposed energy models to predict and understand how input parameters (% CPU utilisation rate, frequency, etc) influence the system and provided solutions to measure precisely the power consumption of single virtual resources.

5. European commission, Cloud Service Level Agreement Standardisation Guidelines, 2014, https ://digital-strategy.ec.europa.eu/en/news/cloud-service-level-agreement-standardisation-guidelines

In data centers or cloud environments, the server power consumption can be divided into static and dynamic parts. The static part (which does not vary with workload) represents the energy consumed by a server when it is idle, while the dynamic cost depends on the current usage. Orgerie et al. [87] define a power-aware model to estimate the dynamic part of energy cost for a VM of a given size, this model keeps the philosophy of the pay as you go model but it is based on energy consumption.

In addition, Benoit et al. [25] review and derive several models under different assumptions (idle, turning on/off costs, time, energy, etc.). Considering On/Off policies is well-known in the literature but can be highly different if one consider that there exists execution time or energy costs to activate or deactivate resources. They show *sequence aware* models that can be based on time or energy constraints. In each model is respectively integrated the turn-off time of VM and the energy cost for switching off. Finally, Zhou et al. [155] also presents a fine-grained energy consumption model and analyses its efficiency in energy consumption of data centers. Their goal is to reduce energy consumption while meeting the quality of service.

With energy measurements and models, cloud providers or customers have the necessary metrics and models to take as inputs for energy-efficient policies. The following treats about how to manage the energy and find efficient scaling policies in cloud environments.

2.2.3 Dynamic resource allocation

Dynamic resource allocation is one of the many technical terms that have emerged in the cloud paradigm. This technique [14] is a very efficient solution in data center owners for adapting resource provisioning to a variable service demand, by setting up activation and deactivation of resources (physical and virtual) according to the workload [25].

As the static part represents a high part of the overall energy consumed by the server nodes, therefore, shutting unused physical resources that are idle leads to non-negligible energy savings. Two main approaches of physical server resource management have been proposed to improve the energy efficiency : shutdown or switching on servers or VMs [121], [117] which is referred as dynamic power management [25], and scaling of the CPU frequency and the voltage referred as Dynamic Voltage and Frequency Scaling (DVFS) [86]. Shutdown strategies (considered in this thesis) are often combined with consolidation algorithms that gather the load on few servers to favour the shutdown of the others [76]. So, managing energy by switching on or switching off resources is an intuitive and widespread manner to save energy. It is also important for virtual resources (VMs or Containers) as the consumption of a physical computer will diminish when virtual resources (or vCPUs) are turned Off, by reducing the physical CPU usage.

Last, as quoted in [25], coarse techniques of shutdown are, most often, not the appropriate solution to achieve energy reduction. Indeed, shutdown policies suffer from energy and time losses when switching Off and On takes longer than the idle period. As well, it may be more efficient to leave some resources On to accept incoming requests.

Moreover, finding the policy that tailors resources to demand is a crucial point that requires accurate assessment of both the energy expended and the performance of the system. Unfortunately, these two measures are inversely proportional, which motivates researchers to evaluate them simultaneously via a unique global cost function. Dynamic resource allocation allows you to quickly increase or decrease your capacity for CPU usage and memory by adding or removing unneeded virtual servers without any downtime. For the activation however there is a difference between containers and virtual machines since the startup time of VMs can be long [101] compared to Docker containers which have a very short start [65].

There are two primary types of scheduling in cloud computing : fixed and dynamic. Fixed scheduling is the more traditional type, where you have to tell your IT provider upfront how much capacity you want. This type of schedule is great for businesses that need a specific amount of resources, but it limits your ability to quickly

grow or decrease your need. Dynamic scheduling, on the other hand, allows businesses to adjust their capacity as needed without any downtime. This is ideal for those who may not know exactly how much capacity they'll need at certain points in time or those who want an easier way to scale up and down as needed (e.g., e-commerce companies). The adjustment of capacity can be done with specific rules such as auto-scaling policies.

2.3 Auto-scaling policies

Auto-scaling is a way to automatically scale the computing resources of your application based on the load in a cloud system. The principal idea is to scale up the resources when there is a spike or rise in web traffic and scale down when traffic levels are low. Some of the world's most popular websites, such as Netflix, have chosen auto-scaling support to meet the rising and ever-changing consumer needs and demands. Amazon Web Services (AWS) [6], Microsoft Azure [7], and Oracle Cloud [8] are some of the most popular cloud computing vendors offering auto-scaling services. For example, AWS handles multiple services, namely AWS service and Amazon EC2 [14]. Cloud users have the option to set the instance count manually or let EC2 do it automatically. Its scaling policies are using their in-house monitoring system Amazon CloudWatch. With AWS Auto Scaling, users can set target utilisation levels for many resources from one intuitive interface and can design scaling plans that automate how groups of different resources respond to changes in demand. Therefore they can optimise availability of resources, costs, or the balance between the two. AWS Auto Scaling automatically creates all the scaling policies and sets the targets for the users based on their preferences. Then, AWS Auto Scaling monitors the users application and automatically adds or removes capacity to the resource groups in real time, tracking changes in demand. As a second example, Azure provides its users a console to set auto-scale programs. They can just navigate to the auto-scale options on their console, add new settings and rules for scaling on various server parameters.

This technology is highly relevant today as the world is committing to reduce carbon emissions and their footprint on the planet. The process helps conserve energy by putting the idle servers to sleep when the load is low. It is most beneficial for applications where the load is unpredictable because it promotes better server uptime and utilisation. This saves electricity and usage bills, as many cloud providers charge based on server usage. In practice, when a user sends a request, the request passes over the internet to a load balancer which communicates to the servers whether to increase or decrease its additional units. Thus many auto-scaling policies are rules taking as inputs the number of requests in the system. Two families of auto-scaling solutions exist : reactive and predictive (or scheduling). Reactive auto-scaling bases its operation on thresholds (see section 2.3.2) specified by the administrator, which activates additional servers when crossed. Thresholds can be set for key server performance metrics such as the percentage occupied capacity. On the other hand, predictive auto-scaling plans the automatic activation of additional servers during traffic peaks based on the time of day. This type of auto-scaling uses Artificial Intelligence (AI) to predict when traffic will be high and schedules server increases in advance.

Nevertheless, finding the best auto-scaling policies can be difficult, especially on large server clusters with massive amounts of information. The principal issue is the search for information (e.g. queuing statistics such as arrival rates) that becomes difficult with millions of users and wide variation in the use of resources. Therefore this prevents to calculate easily the auto-scaling rules. Reinforcement learning, mainly treated in this thesis, is applied because of this lack of information by providing solutions that can learn auto-scaling rules in environments where we don't know all the elements. Last, auto-scaling can be applied horizontally (add virtual or physical resources) and vertically (increase RAM capacity, CPUs, etc.). This thesis will only consider horizontal scaling by adding or removing virtual resources to a pool of machines.

6. https ://aws.amazon.com/fr/autoscaling/
7. https ://azure.microsoft.com/fr-fr/features/autoscale/
8. https ://blogs.oracle.com/developers/post/autoscaling-your-workload-on-oracle-cloud-infrastructure

Several works gave overview about auto-scaling in the cloud [116], [130]. Moreover, Lorido et al. [95] classifies several techniques to find auto-scaling policies :

1. Queuing theory (section 2.3.1) : providing analysis of queuing models to derive when to scale up or down;
2. Threshold-based rules (section 2.3.2) : providing solutions to compute thresholds telling when to scale up or down the number of resources;
3. Reinforcement learning (section 2.3.3) : providing RL methods to compute optimal scaling policies.

The authors also quote control theory and time series analysis as common methods yet we will only show the three main techniques that are considered in this thesis.

2.3.1 Queuing models and control management

Queuing theory is a well-known tool to model cloud systems and to assess algorithms policies by their performances (energy, QoS, etc.). As seen earlier in Chapter II, auto-scaling policies have often been studied with queuing systems. This section presents in more details queuing approaches for cloud environments.

2.3.1.1 Modeling the cloud with queues

In the last decade, many works study resource allocation in the cloud and most of them approaches the problem with queuing models [49]. This allows to represent complex cloud structures (number of nodes, network, arrivals of requests, services rates, etc.), to evaluate performances metrics and is an efficient tool for control management. Both simple queuing models and more complex one have been widely used to analyse the performance in cloud systems.

Gupta et al. [53] published a survey for queuing models in cloud services management. General representation is the following : Requests are generated at the input source by users who require services from the resources (physical or virtual machines), the rate of arrival of request at the service system is determined by the arrival distribution. Various rules also exist for the selection of requests from the queue known as queue discipline or order. Last, many services disciplines exist to represent the variability of applications. So all queuing models differ from their assumptions about the environment : arrivals distributions, services distributions, number of servers, buffer capacity, load balancing process, etc. They allow to analyse important cloud metrics such as expected waiting time, throughput, mean queue length, etc. Many works consider Markovian (Poisson processes) assumptions for arrivals and services distributions : Evangelin et al. [41] consider $M/M/1$ models to represent queuing systems with Markovian arrivals and services and a single resource. Other works consider multi-server systems with $M/M/C$ systems [40, 52, 89]. Some of them [31, 107] study $M/G/S$ models with more general services distributions, and even more complex queues considering also general arrivals distributions such as $G/G/n$ in [12]. Such models can be solved to compute the required resources to process a given input workload, or the mean response time for requests, given a selection of machines.

2.3.1.2 Multi-tier queuing representations

Multi-tier applications can be studied using one or more queues per tier. For example, Bacigalupo et al [19] considered a queuing network of three tiers, solving each tier to compute the mean response time. In addition, Liu et al. [93] study queuing modelling for 3-tier cloud architectures. Queuing networks can also be used to model elastic applications, representing virtual resources as separate queues. For example, Urgaonkar et al. [144] use a network of $G/G/1$ queues per machine. They use histograms to predict the peak workload and uses this information and the model to calculate number of servers per application tier.

In this document, all chapters after and including Chapter V deal with multi-tier cloud infrastructures represented with multi-servers queues in tandem. These systems are little studied in the literature and our work is a real contribution to consider auto-scaling policies in such infrastructures.

2.3.1.3 Server farms queuing representations

In order to represent the problems with activations and deactivations of virtual machines, server farm models have also been proposed [10, 45]. A server farm is a group of computers serving various computing and storage needs from a single location. Such infrastructures have been modelled in the last years with queuing systems. Usually, these server farm models are modeled with multi-server queueing systems [16, 17]. For example, Mitrani et al. [103] uses fluid approximation to compute the activation thresholds in a server farm. Although server farms with multi-server queues allow a fairly fine representation of the dynamicity induced by virtualisation, these models do not address issues related neither to the internal network nor to the VM placement in it. All VMs are instead considered as parallel resources. This makes these models suitable for studying simple nodes of several servers in the cloud.

2.3.2 Threshold-based and hysteresis-based policies

If queuing models allow us to easily compute performance metrics, the decision making for switching on or switching off the VM requires an additional step which remains a key point. The computation of the optimal actions has led to a large field of researches and methods. Dynamic control and especially Markov Decision Processes appear to be the most direct method and more recently Reinforcement Learning (see 2.3.3).

2.3.2.1 Threshold policies

Threshold and *Hysteresis* rules define when to activate or deactivate resources given the load in the system. When a threshold is crossed from above it requires to activate resources to satisfy the demand (and SLA) and when it is crossed from below it decides to deactivate resources to save energy. To quote just a few works that proposed threshold-based auto-scaling policies we refer to the survey [95] which presents many of them. As a matter of fact, [82], [32], [48] study such rules, but differ with chosen metrics, monitoring, workloads data. Moreover, reactive auto-scaling techniques have been applied by Chieu et al. in [33]. Notice also that AwS auto-scaling policies are actually threshold policies [9] that the users can define under their desired conditions and metrics. The policy defines the action to take when the associated CloudWatch alarm is in an ALARM state, in other words, when a certain Key Performance Indicator (KPI) threshold has been crossed by several metrics.

However, it is a major issue to understand if these threshold rules are relevant in practice.

2.3.2.2 Hysteresis policies

On the other hand, threshold policies can imply frequent oscillations if activation and deactivation thresholds are very closed (or equal in some cases) requiring the system to activate and deactivate very often resources. The *hysteresis* is the difference between the two thresholds on a "forward" and "reverse" phase and *hysteresis policies*, which are derived from threshold policies were built to avoid this frequent oscillation. These policies are clearly defined and studied in Chapter IV.

In hysteresis policies for queuing systems [18, 111], servers are powered up when the number of jobs in the system is sufficiently high and are powered down when that number is sufficiently low. More precisely, activations and deactivations of servers are ruled by sequences of different forward and reverse thresholds. The concept of

9. https ://aws.amazon.com/fr/blogs/mt/create-amazon-ec2-auto-scaling-policy-memory-utilization-metric-windows/

hysteresis can also be applied to the control of service rates. They allow not to activate and deactivate the servers too frequently when the load is varying. This makes it possible to correctly adapt variable resources according to demand by means of thresholds [78, 98]. Therefore, multi-server queueing systems working with thresholds-based policies and verifying hysteresis properties have been suggested to efficiently manage the number of active VMs [127, 141]. More recently, Dutreilh et al. [39] stated that thresholds need to be carefully tuned to avoid frequent oscillations in the system, thus considering hysteresis properties.

The other advantage that makes hysteresis policies so appealing is their ease of implementation. This is why, they are a key component of the auto-scaling systems for the widespread cloud architectures Kubernetes for docker components, and Azure or Amazon Web Services [14] for virtual machines. Henceforth, there is a great interest of studying the computation of hysteresis policy since threshold values can be plugged into auto-scaling systems that are implemented in the major cloud architectures. Researches on hysteresis policies are twofold : first exploration of the conditions that insure the optimality of hysteresis policy. The second axis studies the computation of the threshold values.

2.3.2.3 Optimality of threshold and hysteresis rules

It exists in the literature several works studying the structure of the optimal policy in queuing systems [62, 81], network and finally Cloud. Federgruen et al. [42] show optimality of threshold policies in single-server queuing systems with server vacations. Yang et al. [150] also study structural properties of the optimal resource allocation policy for single-queue systems. The optimal number of servers follows a step function following a bang-bang control policy. Yang et al. provide conditions under which the bang-bang control policy takes place. In optimal control problems, it often happens that a control is bounded by a lower and an upper bound. If the optimal control goes from one extreme to the other (i.e., it is strictly never between the two bounds), it is called a bang-bang control policy. Also in [94] is shown that under some assumptions the optimal policy in a tandem queue system is a bang-bang control policy with monotonicity properties.

For hysteresis rules, it appeared very early in the work of Bell for a $M/M/2$ queuing model [137], that the optimal policy in such models has a special form and is called hysteresis. In addition, Szarkowicz et al. [134] showed the optimality of hysteresis policies in a $M/M/S$ queuing model with a control of the vacations of the servers. For models with control of the service rates, the proofs were made in Hipp [58] in the 80s and in Serfozo [78] for M/M/1 queues in the 90s.

2.3.2.4 Computation of threshold and hysteresis rules

Now, creating the rules requires an effort from the clients (or cloud providers), who need to select the suitable performance metrics and need to define carefully thresholds. Moreover it is stated in [95] that it is difficult to select the corresponding thresholds in real systems. Indeed, the calculation of effective threshold values first encounter the same difficulties as single threshold policies (base-stock policies).

For hysteresis models, the calculation of optimal thresholds received few attention in the past and two major trends appeared. The computation of the optimal policy can be done by means of adapted usual dynamic programming algorithms in which the structured policies properties are plugged. A similar treatment has been addressed for routing in [110]. The alternate way is similar to the single threshold research for base-stock policies which is very common in inventory management. A local search in the works [84, 118] is used to explore the optimal thresholds. The computation of expected measures associated with a set of thresholds is complex [28]. It requires the computation of the stationary distribution either by standard numerical methods see e.g. [149] or after a Markovian analysis with simpler and faster computations (e.g. [146] uses iterative methods to compute the stationary distribution). As cloud systems are modelled by multi-dimensional systems, defined on very large state spaces, then the stationary distribution computation is difficult. Fortunately, the computation of the per-

formance measures of hysteresis multi-server systems has been already studied in the literature. Different efficient resolution methods have been developed. Among the most significant works, we quote the work of Ibe and Keilson [63] refined in Lui and Golubchik [98]. Both solve the model by partitioning the state space in disjoint sets to aggregate the Markov chain. Exhaustive comparisons of the resolution methods are made in [71] : closed-form solution of the stationary distribution, partition of the state space, and matrix geometric methods applied on QBD (Quasi birth and death) processes are studied.

In [6], Song et al. claims that, for single threshold models, the second approach dealing with stationary distribution computations are generally more effective than Markov Decision Process approaches. Such a comparison has not been performed yet for hysteresis especially since few works implement a MDP algorithm with a structured policy. Chapter IV proposes and compares two mathematical approaches to compute optimal thresholds and hysteresis policies under specific assumptions (also see [139]).

2.3.3 Reinforcement Learning for cloud resource allocation

Numerous works have been devoted to compute optimal policy in multi-server queue models with MDP (see [139] and references therein). However, Markov Decision Process framework requires a perfect knowledge of the model (queuing statistics such as arrival or service distributions etc.). Unhappily, these values are not always known in practice and Reinforcement Learning techniques should be applied to dynamic resource allocation to overcome this lack of information. Without any a priori knowledge, RL methods can determine the best scaling decision to take in each state, given the input workload. This section first displays MDP approaches where knowledge of the environment is given to the agent, then auto-scaling RL solutions [46] where environments information are unknown (statistics, users, etc.).

2.3.3.1 Markov Decision Process approaches

Markov Decision Process has been widely used to model resource management problems including auto-scaling problems. Different algorithms exist to find the optimal management policy, when the underlying transition probabilities are known [139]. Since the seminal work of Mc Gill in 1969 (with an optimal control model) and that of Lippman (with a Markov decision process model) numerous works have been devoted to similar multi-server queue models using Markov decision processes (see [137] and references therein for the oldest, and more recently [151] and [90] to quote just a few). Moreover, [91] is a recent survey about MDP approaches for queues and networks. It reviews all MDP applications for control in queuing and networking systems. For example, Okamura et al. [112] study MDP approaches for dynamic power management problems in order to save power consumption. They consider an optimal power-aware design in a cloud system represented by a cluster. Unfortunately, not all of them received rigorous treatment and the study of unichain or multichain property is often ignored.

Now, these works require full knowledge of the environment which is not obvious in real cloud systems. To overcome this issue and deal with missing information about environments, reinforcement learning methods have been applied.

2.3.3.2 Reinforcement learning approaches

Again, two surveys [46, 95] address reinforcement learning for auto-scaling policies in cloud environments. Many works for RL application on cloud resource allocation problems are discussed and different taxonomies about the resolution techniques, the criteria to optimise as well as the type of problem are presented. As mentioned earlier, RL techniques can learn the optimal auto-scaling policy without requiring the knowledge of the statistics of the system : either it learns from experience in a model-free fashion or the agent aims to learn this

knowledge in a model-based fashion. Following [46], we decompose this paragraph in two parts : model-free RL applications and model-based applications.

Model-free techniques : Most of the RL techniques applied to resource allocation in the cloud are model-free and the vast majority of them are standard Q-learning and Deep Q-learning techniques. For example, a model-free reinforcement learning technique with Q-Learning for autonomic resource allocation in the cloud was proposed by Dutreilh et al. in [38]. In this work an agent has to control the number of resources and optimise a cost function that takes into account virtual machines costs and SLA. They focus on a single physical node hosting several virtual resources and consider the $(w; u; p)$ state definition, where p stands for performance in terms of average response time to requests, w for workload and u the number of virtual servers. An other example is the work from Tesauro et al. [138]. They propose model-free RL using $(w; u_{t-1}; u_t)$ state representation, where w is the total number of user requests observed per time period; u_t and u_{t-1} are the number of activated VMs in the current time step, and the previous time step, respectively. This representation will be used along the thesis (i.e. considering number of requests and VMs at each time step).

For our problem, Jin et al. [68] propose Q-learning to derive auto-scaling policies in the cloud to minimise delays and number of activated resources and consider again a single physical node. To quote more, the following works focus mainly on Q-learning algorithm on a single physical node but have different optimisation criteria or state space : the work [61] considers the response time and the average resource utilization and [148] proposes an approach based on inhomogeneous VM. More recently and in modern networks, works have been presenting RL solutions [77] for 5G network slicing and [80, 124] for slicing with Deep RL, to quote just a few.

However, many works raise performance issues with model-free RL techniques. Indeed, the problem of poor performances in the early steps has been addressed in a number of ways. The principal problem is the lack of initial information and the randomness of the exploration policy. For this reason, Dutreilh et al. [39] proposed a custom heuristic to guide the state space exploration. They also propose an initial approximation of the Q-function that updates the value for all states at each iteration, and also speeds up the convergence to the optimal policy. Other works have also attempted to reduce this training time. It is addressed with an exploration policy that visits several states at each step [119] or using parallel learning agents [22]. In the latter, learning agents do not need to visit every state and action since they can learn the value of non-visited states from neighboring agents.

Overall, many works show that while model-free RL techniques hold great promise for learning adaptive control policies, it still suffers from slow convergence and detrimental random exploration. So this appeal to study new paradigm of RL techniques to overcome these issues.

Model-based techniques : Although researchers have attempted to improve model-free techniques, very few of them have tried to apply model-based RL approaches. In this context, model-based reinforcement learning techniques [106] can decrease exploration steps by learning a model of the environment, allowing the agent to update faster the Q-Value by a supplementary planning phase. Nevertheless, [46] mentions only two works that are classified as using model-based techniques for Cloud auto-scaling scenarios and these two works ought to estimate the transition probabilities and thereafter solve the problem with MDP algorithms. Hence, due to the requirement of having a complete model of the environment in model-based methods, [21] estimates the probability distribution of the transition between states by counting occurrences of visits in state-action pairs (s, a) and (s, a, s') and the planning of the policy is done in an offline mode, by comparing MDP resolution algorithms with Genetic Algorithm. However, as quoted in [106], the model based framework is larger than the simple models presented in [46]. This, coupled with the very few amount of works in literature about model-based reinforcement learning for cloud applications call for further research in this topic and for comparison of model-free algorithms.

Therefore, there is a great interest to study model-based techniques and compared them with state of the art model-free RL methods. This is the main application in Chapter V. On the other hand, it could be extremely valuable to devise methods that limit the dimensionality while offering more guarantees of accuracy and greater explainability. This is the goal of the last two chapters which improves model-based RL methods by considering

state variables dependencies as a knowledge for the agent. Hence, Chapter VI deals with factored representation and Chapter VII deals with causal representation.

CHAPTER 3

REINFORCEMENT LEARNING AND NETWORKING APPLICATIONS

This chapter introduces formalism for reinforcement learning which will be the main framework throughout the manuscript. It covers the mathematical formalism of Markov Decision Process then present a global taxonomy of RL techniques from model-free to model-based. It also introduces the factored and causal approaches. Last, it presents an industrial application with state of the art model-free applied to combinatorial optimisation problem : Deep-Q-Network.

Reinforcement learning [8] is concerned with how a software agent ought to take actions in an environment to maximise some cumulative reward. The autonomous agent must discover which decisions will produce the greatest long-term benefit by trial-and-error search. Markov Decision Process [5] is the theoretical model behind this framework. It allows to describe the interactions between the agent and its environment in terms of states, actions, and rewards (see Figure 3.1). The basic idea is to represent aspects of a real problem faced by a learning agent that interacts over time with its environment to achieve a goal.

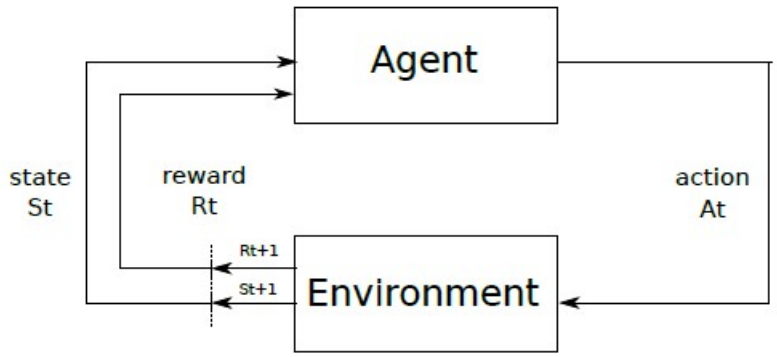


FIGURE 3.1 – RL interaction between an agent and the environment (Figure from [46])

3.1 Markov Decision Process

3.1.1 Elements

Markov Decision Process, named after Andrei Andreyevich Markov (1856-1922), is an extension of Markov chains with supplementary elements : actions and rewards. A MDP is a mathematical model for the random evolution of a memoryless system. The classical model of a MDP is defined as a 5-tuple $(\mathcal{S}; \mathcal{A}; \mathcal{P}; \mathcal{R}; \gamma)$ where :

* $\mathcal{S}$ represents the environment state space : it can be seen as a stochastic process that evolves over time $\{\mathcal{S}^t\}_{t\in\mathbb{N}}$ depending on the system's randomness and the agent's actions. We denote by a lowercase s the value of the state.
* $\mathcal{A}$ represents the action space : it is the set of possible actions the agent can chose in a given state $s \in \mathcal{S}$ and that will influence environment state. We denote by a lowercase a the value of the action.
* $\mathcal{P} : \mathcal{S} \times \mathcal{S} \to [0, 1]$ with $\mathcal{P}_a(s, s') = \mathcal{P}(s^{t+1} = s'|s^t = s; a^t = a)$ represents the probability that action a in state s at time t will lead to state s' at time $t + 1$: it describes the system's randomness;
* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with $\mathcal{R}(s, a)$ represents the immediate reward received after taking action a in state s : it describes the goodness of agent's decision a in state s. We denote by capital $\mathcal{R}$ the reward function and by a lowercase r the reward value.
* $\gamma \in [0, 1]$ is the discount factor : it represents how much the agent care about the future.

Such systems respect the Markov property, i.e. that a given future state depends only on the present state and having information about the past does not bring supplementary knowledge. In probabilistic words, the stochastic process $\{\mathcal{S}_t\}_{t\in\mathbb{N}}$ satisfy the Markov property if :

$$p(\mathcal{S}^t = s^t|\mathcal{S}^{t-1} = s^{t-1}, \ldots, \mathcal{S}^0 = s^0, a^{t-1} = a) = p(\mathcal{S}^t = s^t|\mathcal{S}^{t-1} = s^{t-1}, a^{t-1} = a)$$

Now, to evaluate its performance in the environment, there are three fundamental elements that the learning agent carries : the policy q, the state-value function $\mathcal{V}$ or action-value function Q, and optionally, the model of the environment.

* The policy $q : \mathcal{S} \to \mathcal{A}$
* The state-value function $V_q : \mathcal{S} \to \mathbb{R}$ or the action-value function $Q_q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
* An environment model $\{\mathcal{P}, \mathcal{R}\}$.

To summarise : The state space $\mathcal{S}$ represents the intrinsic state of the environment. Yet it happens that the agent can only observe a part of the state and we will refer to the observation space, denoted $\mathcal{O}$ where $\mathcal{O} \subseteq \mathcal{S}$. The action space $\mathcal{A}$ is the set of available decisions for the agent. The *reward* $\mathcal{R}$ is a numerical signal that tells the agent if its decision is good or not and the agent's goal is to maximise the total amount of received reward. For this aim, the agent needs to determine the best policy q. The policy q is a function defining what decision a the agent should take in a given state s. In general, the agent attempts to maximise the return $\mathcal{G}$ that is defined as a specific function of the reward sequence. The return is defined as the sum of the discounted rewards :

$$\mathcal{G} = E_{a^t\sim q}[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s^t, a^t)]$$

where T is a final time step. The reward signal is the primary basis for altering the policy q. If an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward signals may be stochastic functions of the state of the environment and the actions taken, yet we consider in this thesis only deterministic rewards.

Whereas the reward signal indicates what is good in an immediate sense, a *value function* $V \in \mathcal{V}$ specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. The state-value function V_q is the expected cumulative reward over the future, when agent follows a specific policy q. It represents an approximation of the true return $\mathcal{G}$. Thus the autonomous agent aims to maximise this value function and its best policy q will be to reach states in which the value function is the highest. The action-value function Q_q represents the same metric but integrates the effect of the first chosen action a. It represents for the agent the goodness of taking action action a in state s and how good it is to be in this state. We have :

$$V_q(s) = \mathrm{E}_q\Big(\sum_{t=0}^{T} \gamma^t \mathcal{R}(s^t, q(s^t)) \mid s^0 = s\Big) .$$

We also define the action-value function (also called Q-function) :

$$Q_q(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in S} P(s'|(s,a)) V^q(s') .$$

and provide relations between V, Q and q :

$$V_q(s) = \text{argmax}_a Q_q(s,a) \text{ and } q^* = \text{argmax}_q V_q$$

3.1.2 Markov Decision Process resolution

In the Markov Decision Process framework [5], we have a full information about transition and reward functions. This allows the software agent to do planning and iteratively update its policy or value function as it could imagine the consequences of its actions. Before presenting the MDP algorithms, we provide detailed characteristics about MDPs [2].

3.1.2.1 Criteria, policies and optimality

Policies types

Formally, policies are rules that tell the agent what action it should take given an observation of the environment. This observation can be the history of observations [2] $h^t = (s^0, a^0, s^1, a^1 \ldots, s^t)$ in which the system is *History-Dependent* or it can be the last observation of the system s^t in which the system is *Markovian*. Next, we specify that the policy q can be *deterministic* or *stochastic*. In the first case, the policy is defined by $q : S \to \mathcal{A}$, telling the agent what action a it should act when it observes state s. The stochastic case where the policy is defined by $q : S \times \mathcal{A} \to [0,1]$ indicates the probability for the agent to play an action a in state s. Moreover, we define stationary policies of the form : $\forall t,\ q^t = q$, i.e. the policy does not evolve over time. We characterise four families of policies :

* HS for stochastic history-dependent policies;
* HD for deterministic history-dependent policies;
* MA for stochastic Markovian policies;
* MD for deterministic Markovian policies;

Performance criteria and optimality When a software agent solves a MDP, it can consider several criteria depending on the optimisation problem. We define 4 different types of performance criteria with their associated value function. $\forall s \in S$ and for a determined policy q :

* Finite Horizon Criteria with : $V_q^N(s) = E^q[\sum_{t=0}^{N-1} \mathcal{R}_t \mid s = s_0]$

* Discounted Criteria : $V_q(s) = E_q[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_t \mid s = s_0]$

* Total Criteria : $V_q(s) = E_q[\sum_{t=0}^{\infty} \mathcal{R}_t \mid s = s_0]$ with $\gamma = 1$

* Average Criteria : $\rho^q(s) = \lim_{n\to\infty} E^q[\frac{1}{n}\sum_{t=0}^{n-1} \mathcal{R}_t \mid s = s_0]$

In the MDP setting, the optimal policy q^* and its associated optimal value V^* or ρ^*, refers to the policy that has the best value for all state, i.e. : q^* is optimal if $\forall s \in S\ ,\ V^*(s) \geq V(s)\ ,\ \forall V \in \mathcal{V}$ where $\mathcal{V}$ is the set of value functions and $q^*(s) = \text{argmax}_{a\in\mathcal{A}} V(s)$ [1]. Now it has been proven that optimal policies in MDPs are Markovian policies [2], i.e. that it exists a q^* that belongs to MA. And depending on the chosen criteria, the optimal policy can be stochastic or deterministic. In this thesis, we consider only two criteria : discounted criteria and average criteria.

Optimality and Bellman Equations Solving MDP implies to separate the expected sum of rewards which means to maximise or minimise the instantaneous reward generated by the selected action a in present state s plus the future rewards from future possible states s'. We define for general cases the *Bellman dynamic programming operator* $\mathcal{T} : \mathcal{V} \longrightarrow \mathcal{V}$ s.t.

$$\mathcal{T}V(s) = \max_{a\in\mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma . \sum_{s'\in S} p(s'|s,a).V(s') \right\}$$

In matricial form :

$$\mathcal{T}_q V = \mathcal{R}_q + \gamma P_q V$$

We can derive from these equations the *Bellman equation* used it MDP resolution algorithms.

$$V(s) = \max_a \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a)V(s') \tag{3.1}$$

3.1.2.2 Dynamic Programming Algorithms

The term *dynamic programming* (DP) refers to the decomposition principle of the Bellman equation, which from this equation gives a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a MDP. We refer to state of the art DP algorithms as : Value Iteration (VI), Policy Iteration (PI). These methods are also called *planning* methods since they allow the agent to plan by evaluating value function and policy with the model $\mathcal{M}$ from the recursive *Bellman equation* 3.1. We first present the two algorithms for the discounted criteria.

Value Iteration VI is obtained simply by turning the Bellman optimality equation 3.1 into an update rule. It learns an optimal value function V^* by solving iteratively the Bellman equation and derives from it the optimal policy q^*. It iteratively updates the value function by solving for all states $s \in \mathcal{S}$ the equation 3.1. Intuitively, the agent updates its value function at time step $t+1$ by integrating the immediate reward $\mathcal{R}(s,a)$ plus the value function of potential next states s' aggregated by the probability to reach s' under state s and action a. After convergence, it derives the policy by taking the best action in each state s, thus maximising the value function V.

1. in a Max optimisation problem

Algorithm 1: Value Iteration - Discounted criteria

Input: V^0, q^0, ζ accuracy
Output: q^*, V^*
Data: $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$ is known

repeat
  for $s \in \mathcal{S}$ **do**
    $V^{k+1}(s) = \max_a \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^k(s')$ `// Bellman equations`
until $||V^{k+1} - V^k|| \leq \zeta$
`/* Policy derivation */`
for $s \in \mathcal{S}$ **do**
  $q^*(s) = \text{argmax}_a \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^k(s')$

Policy Iteration The Policy Iteration algorithm focuses on the policy q. It is decomposed into two steps : Policy Evaluation (PE) where the algorithm evaluates the policy by calculating the value function V under current policy q and Policy Improvement (PI) by taking the best actions that maximises the value function V.

Algorithm 2: Policy Iteration - Discounted criteria

Input: V^0, q^0
Output: q^*
Data: $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$ is known

repeat
  `/* Policy Evaluation */`
  for $s \in \mathcal{S}$ **do**
    $V^{k+1}(s) = \mathcal{R}(s, q^k(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, q^k(s)) V^k(s')$
  `/* Policy Improvement */`
  for $s \in \mathcal{S}$ **do**
    $q^{k+1}(s) = \text{argmax}_a \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^k(s')$
until $q^k(s) = q^{k+1}(s)$

Relative Value Iteration Finally we present the *Relative Value Iteration* (RVI) algorithm that solves the average criteria in pseudo-code 3. The method does not deal with the usual value function V, but works with the *relative value function* U that is computed by :

$$\forall s \in S, U_q(s) = E_q[\sum_{t=0}^{\infty} \mathcal{R}_t - \rho_q \,|\, s = s_0]$$

where ρ_q is the average gain under policy q. More details are provided for the study of average criteria in Chapter IV.

Algorithm 3: Relative Value Iteration - Average criteria

Input: $U^0 = 0, q^0, \zeta$ accuracy
Output: q^*, U^*, ρ^*
Data: $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$ is known

repeat
  Choose $s_0 \in \mathcal{S}$
  $\rho^{k+1} = \max_a \mathcal{R}(s_0, a) + \sum_{s' \in \mathcal{S}} p(s'|s_0, a) U^k(s')$ `// Average gain calculation`
  for $s \in \mathcal{S}$ **do**
    $U^{k+1}(s) = \max_a \left[\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) U^k(s')\right] - \rho^{k+1}$ `// Average Bellman equations`
until $||U^{k+1} - U^k|| < \zeta$
`/* Policy derivation */`
for $s \in \mathcal{S}$ **do**
  $q^*(s) = \text{argmax}_a \left[\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) U^k(s')\right]$

3.2 Reinforcement learning

Let us define the environment model $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$. It is an object that mimics the dynamics behaviour of the environment. Given a state s and an action a, it predicts the next state s' with transition distribution $\mathcal{P}(s, a, .)$ and returns the reward $\mathcal{R}(s, a)$. If the model $\mathcal{M}$ is perfectly known, one can solve the problem by learning an optimal policy with classical MDP solutions (see section 3.1.2). On the contrary, if $\mathcal{M}$ is unknown, the problem can be solved with reinforcement learning techniques that will be explain in this section. Unlike Markov Decision Process resolution with dynamic programming, the RL agent have to learn the optimal policy or value function by interacting with the environment in trial-and-error mode. In other words, the RL agent can not perform planning computations and needs to collect data from environment interactions to update its policy or value function. The field of reinforcement learning can be decomposed into several branches that are displayed in section 3.2.2 which provides the different classes of algorithms. We first provide the main intuition of reinforcement learning algorithms, i.e. how the software agent can learn with collected experiences.

3.2.1 Main idea of Reinforcement Learning

The first idea borrowed from reinforcement learning techniques is the *Monte Carlo* methods, where the goal is to perform experiences to estimates values. The goal in the RL paradigm would be to estimate the value function V^q of a fixed policy q with simulated trajectories and updating the value at the end of each trajectory. In a nutshell, the agent observes environment states s and decide to perform an action a given by its policy $q(s)$. From this interaction, the agent collects a reward r and observes the new environment state s'. In a simulation, it would collect a history of samples (s, a, r, s') that it could use to learn the optimal value function or policy.

In a Monte Carlo simulation for RL, the agent will simulate trajectories k from each state s for a fixed period T following policy q and will look at the cumulative sum of reward (or Return) obtained from that trajectory $R_k(s)$. Then, it can approximate the value function V as :

$$\forall s \in S\ ,\ V_{k+1}(s)\ =\ \frac{R_1(s) + R_2(s) + \ldots + R_{k+1}(s)}{k+1} \tag{3.2}$$

The essence of this method has led to a more modern approach called *Temporal Difference* (TD)-learning, which is the principal methodology for *Q-Learning* algorithm. If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be TD-learning. This method is a combination of Monte Carlo ideas and DP ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome. In other words, TD-learning methods are bootstrapping. We can start from the Monte Carlo method equation to understand the TD-learning process and rewrite 3.2 :

$$\forall s \in S\ ,\ V_{k+1}(s)\ =\ V_k(s) + \frac{1}{k+1}[R_{k+1}(s) - V_k(s)]$$

Which leads us to :

$$\forall s \in S\ ,\ V_{k+1}(s)\ =\ V_k(s) + \alpha[R_{k+1}(s) - V_k(s)] \tag{3.3}$$

where α is the learning rate.

As said, TD-learning uses the incremental characteristics of dynamic programming and the trial characteristic of Monte Carlo. We focus on discounted criteria with γ and present TD(0) or *one-step* TD which considers only one time step. We depict next the reasoning that leads to the TD(0) update equation. If the value functions estimates were exact, i.e. that it estimates perfectly the return G, we would have the following :

$$\begin{aligned} V(s_t)\ &=\ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \\ V(s_{t+1})\ &=\ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \end{aligned}$$

So

$$V(s_{t+1})\ =\ r_t + \gamma V(s_{t+1})$$

This leads to the one-step update equation for the value function, also called the *Widrow-Hoff Equation* :

$$V(s_t)\ =\ V(s_t) + \alpha[\underbrace{r_t + \gamma V(s_{t+1}) - V(s_t)}_{\text{TD-error}}] \tag{3.4}$$

Like in Dynamic Programming, the value $V(s_t)$ is updated with the value of the successor $V(s_{t+1})$ and with the new reward received r_t from the environment (see pseudo-code 4). Obviously, TD methods have an advantage over DP methods in that they do not require a model of the environment, of its reward function and next-state probability distributions. The next most obvious advantage of TD methods over Monte Carlo methods is that they are naturally implemented in an on-line, fully incremental fashion and they do not need to wait until the end of an episode. Moreover, the convergence of TD-learning methods have been proven, i.e. that for any policy

q, TD(0) converges to V_q.

Algorithm 4: TD(0)-Learning - Discounted criteria

Input: V^0, q, α learning rate, γ discounted rate
Output: V_q
Data: $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$ unknown
/* Several simulation episodes */
for episodes **do**
  Initialise $s = s^0$
  /* Several steps until end of episode */
  for iterations **do**
    Select action $a = q(s)$ and collect r and s'
    $V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)]$
    $s \leftarrow s'$

This computational logic is present in most of the RL algorithms, that is the approximation of the return or cumulative sum of rewards which can be seen as a target for the value function or policy to update, in a supervised learning mode.

3.2.2 Taxonomy of RL algorithms

Now, to structure reinforcement learning techniques, we provide in this section an overview of RL algorithms. Our goal here is to highlight the most foundational design choices in RL algorithms about what to learn and how to learn it.

3.2.2.1 Model-free versus Model-based

One of the most important branching points in a RL algorithm is the question of whether the agent has access to (or learns) the model $\mathcal{M}$ of the environment. Algorithms which use a model are called *model-based*, and those that don't are called *model-free*. Model-free techniques aim to update the policy or value function directly from experience without involving a model of the world, whereas the second one, on the contrary, works with the model of the world. In model-based methods, the model can be given which bring us back to MDP resolution, or it can be learned by the agent from experiences. The agent is building an approximated model $\tilde{\mathcal{M}} = \{\tilde{P}, \tilde{\mathcal{R}}\}$ to perform planning, i.e. update value function or policy.

The main advantage of having a model is that it allows the agent to plan by thinking ahead, seeing what would happen for a set of possible choices, and explicitly deciding between its options. Agents can then integrate the results of the planning into a learned policy. A particularly famous example of this approach is AlphaZero [129]. While model-free methods renounce to the potential gains of sampling efficiency associated with using a model, they are easier to implement and tune. In the literature, model-free methods are more popular and have been more widely developed and tested than model-based methods for industrial applications.

3.2.2.2 Value-based versus Policy-based

Another critical branching point in a RL algorithm is the question of what to learn. The list of usual tools includes :

* policies, either stochastic or deterministic
* action-value functions (Q-functions)

* value functions (V-functions)

Learning the policy refers to *policy-based* RL algorithms while learning action-value or value functions refers to *value-based* RL algorithms. This distinction takes up the difference between VI and PI in DP algorithms.

This two orthogonal distinctions are represented in Figure 3.2.

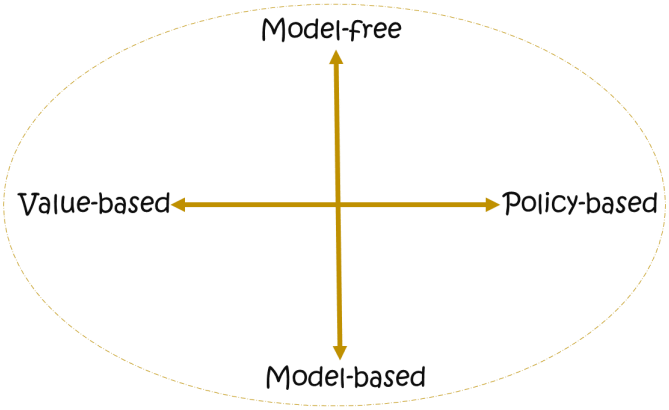


FIGURE 3.2 – Two principal differences between reinforcement learning methods

3.2.3 Model-free reinforcement learning

Since model-free techniques does not work with a model of the environment, they only learn from experiences collected with environment interactions. As developed in section 3.2.1, the goal is to derive the value function or policy from collected data.

3.2.3.1 Value-based (Q-learning)

This section focuses on the learning of the action-value function Q. The concept is borrowed from TD-learning, extended on the Q-value function. The idea is that if the agent can correctly approximate $Q(s, a)$ for all state-action couples (s, a), it can easily derived the best action to take in a state s by computing :

$$a^* = \text{argmax}_a Q(s, a)$$

Tabular Q-Learning We first consider tabular methods where the Q-value function is implemented as a table with actions and states as entries. We display the *Q-learning* [147] algorithm update equation (extended from 3.4) with $Q(s, a)$ instead of V :

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r(s, a) + \gamma \text{max}'_a Q(s', a') - Q(s, a) \right] \tag{3.5}$$

The action selection process in the simulations is an *epsilon-greedy* policy which allows the agent to exploit and explore over time. It usually starts with a high value of epsilon and decrease it over episodes. The process draws a random probability between 0 and 1 and compares it with the value of epsilon ε. If it is below, the agent explores by taking a random decision, if it is above the agent exploit by following its policy, i.e. the best action a that maximises $Q(s, a)$. This is one well-known technique to deal with the *exploration-exploitation* paradigm in model-free reinforcement learning.

We finally provide the pseudo-code in Algorithm 5.

Algorithm 5: Q-Learning

Input: Q^0, q^0
Output: q^*, Q^*
Data: Dynamics unknown
/* Loop until end of episodes or criterion */
for $e \in \{1, \dots, N_e\}$ **do**
  Select state $s \in \mathcal{S}$ // Initial state
  Take action $a \in \mathcal{A}$ with epsilon-greedy policy. // Action selection
  Observe s' and reward $\mathcal{R}(s, a)$ and collect tuple $< s, a, r, s' >$ // Collecting tuples data
  $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r(s, a) + \gamma \max'_a Q(s', a') - Q(s, a)\right]$ // Update Q

Now, we have seen that the Q-learning method has a tabular representation of the Q-function and is efficient on small action and state spaces. This algorithm can be slow converging since we need further update of all $Q(s, a)$ values to converge to the optimal solution. Moreover, at each time step the method only updates $Q(s, a)$ for one couple (s, a). The rise of dimensionality and number of state-actions pairs makes it irrelevant and approximation for the Q-function is needed. This is why Deepmind's team [105] developed an algorithm where the Q-function was approximated by a neural network [1] : *Deep-Q-Network* or *DQN*.

Deep-Q-Network The idea is similar to Q-learning instead it updates the weight of the Q neural network with an experience replay buffer $\mathcal{D}$ that saves all the agent's interactions (s, a, r, s') with the environment. The first gain is that one learning iteration will update the weights of Q and thus will update all $Q(s, a)$. Now the action-value function is a neural network parameterised by weights $\theta : Q_\theta(s, a) = Q(s, a|\theta)$. The objective function to optimise is a loss function computing the TD-error. Basically, the idea is to update the weights of the neural network in a supervised learning mode, by minimising the TD-error over a batch of samples. In other words, for an experience tuple (s, a, r, s'), the algorithm builds the mean squared error loss function :

$$L(s, a) = [y(s, a) - Q(s, a|\theta)]^2$$

where $y(s, a) = r + \gamma \max_a Q_\theta(s', a)$.
More usually, it computes the loss function over a batch of experiments taken from the memory buffer :

$$L = E_{(s,a,r,s') \in \mathcal{D}}[y(s, a) - Q(s, a|\theta)]^2$$

The update of the neural network weights is made with *Gradient Descent* techniques such as Stochastic Gradient Descent or ADAM optimiser, which are often use in practice. For the neural network architecture, two structures exist. In the first one, the Q-network has in the first layer for inputs the state's variables of the system and for the last layer the outputs $Q(s, a)$, i.e. $Q_\theta : S \to \mathbb{R}^{|\mathcal{A}|}$. The second architecture takes as inputs the state and the action a and outputs the unique value of $Q_\theta(s, a)$, i.e. $Q_\theta : S \times \mathcal{A} \to \mathbb{R}$. The choice of the architecture depends usually on the cardinality of $\mathcal{A}$. To avoid correlations between samples for the weights updates, the process randomly select batch of experiments from the buffer replay $\mathcal{D}$ to have $i.i.d$ samples. Finally, to reduce oscillations while learning, the algorithm freezes the target Q-neural network for some learning iterations : $\hat{Q}$. Algorithm 6 displays the pseudo-code for the DQN.

When we refer to value-based reinforcement learning, one often refers to Q-Learning and DQN techniques. Yet, it exists in the literature many improvements [57] (initialisation, multi-agent distributed, exploration-exploitation trade-off, double DQN, Rainbow DQN, etc.) of this technique.

Algorithm 6: Deep-Q-Network

Input: Q_θ action-value function, $\hat{Q}_{\hat{\theta}}$ target value function with $\hat{\theta} = \theta$, $\mathcal{D}$ replay memory, N_e number of episodes
Output: q^*, Q^*
Data: Dynamics unknown

```
/* Loop until end of episodes or criterion                                  */
```

for $e \in \{1, \dots, N_e\}$ **do**
  Initialise s^0
  for $t \in \{1, \dots, T\}$ **do**
    Select $a^t = \text{argmax}_a Q_\theta(s^t, a)$ if exploitation else select randomly a^t with exploration
    Collect s^{t+1}, r^t and store the whole transition (s^t, a^t, r^t, s^{t+1}) in $\mathcal{D}$
  /* Learning   */
  Sample randomly batch of experiences (s, a, r, s') from $\mathcal{D}$
  Set $y = r$ if terminal sample $(t = T)$ else
  Set $y = r + \gamma \text{max}_a \hat{Q}_{\hat{\theta}}(s, a)$
  Update θ with gradient descent techniques on $L = \sum_{samples} \left(y - \hat{Q}_{\hat{\theta}}(s, a)\right)^2$
  Sometimes update the target network $\hat{\theta} = \theta$

3.2.3.2 Policy-based RL

Methods in this family represent a policy explicitly as $q_\theta(a|s)$, i.e. most of the time the policy is stochastic in policy-based (or policy-gradient) methods and we will denote them interchangeably by $q(a|s, \theta)$ or $q_\theta(a|s)$. The policy indicates the probability to select action a in state s when the policy is parameterised by weights θ. One optimises the parameters θ either directly by gradient ascent on the performance objective $L(q_\theta)$, or indirectly, by maximising local approximations of $L(q_\theta)$. Note that the objective function $L(q_\theta)$ varies depending on the policy-based algorithms. This optimisation is almost always performed on-policy, which means that each update only uses data collected while acting according to the most recent version of the policy. Policy optimisation can also involve learning an approximator $V_\phi(s)$ (ϕ being the weights for the value function approximation) for the on-policy value function $V^q(s)$, which gets used in figuring out how to update the policy.

We provide in algorithm 7 the main behavior of policy-based algorithms, by providing the pseudo-code of *REINFORCE*, a Monte-Carlo policy-based algorithm. This algorithm generate sequences of data by following its policy q. With collected rewards, it calculates the return and uses it to update the weights of the policy with

ascent gradient computations.

Algorithm 7: REINFORCE : a policy-based algorithm

Input: $q_\theta(a|s), \theta \in \mathbb{R}^d$, α learning rate, γ discount rate
Output: q^*
Data: $\mathcal{M}$ unknown
/* Repeat until end of episodes or criterion */
for $e \in \{1, \dots, N_e\}$ **do**
  Generate a sequence $s^0, a^0, r^1, s^1, \dots, s^{T-1}, a^{T-1}, r^T$ following $q_{\theta_e}(.|.)$
  for $t \in \{1, ..., T-1\}$ **do**
    $\mathcal{G} \leftarrow$ return at step t
    $\theta \leftarrow \theta + \alpha\gamma^t \mathcal{G} \nabla_{\theta_e} \ln q_{\theta_e}(a^t|s^t)$

However, in this thesis, we only consider value-based methods. Therefore we just recall well-known policy-gradient algorithms without providing details. The main differences between policy-based or policy-gradient methods is how they update the weights, the loss function they consider but also the architecture of the objects to optimise (value function, policy). For example, some of them, *actor-critic* methods such as A2C and A3C [104] update both the policy (actor) and the value function (critic) in a asynchronous manner. Moreover, in *Policy Proximal Optimisation* [123], known as *PPO*, one uses an approximation of the advantage function $A(s,a) = Q(s,a) - V(s)$ and a clip function to update the weights of the policy. The algorithm considers the following surrogate objective function to learn the new policy weights θ' :

$$L = \min\Big(\frac{q_{\theta'}(a|s)}{q_\theta(a|s)} A^{q_\theta}(s,a), \text{clip}\big(\frac{q_{\theta'}(a|s)}{q_\theta(a|s)}, 1-\epsilon, 1+\epsilon\big) A^{q_\theta}(s,a)\Big)$$

that imposes a clip interval on the probability ratio term. The clip function ensures the policy at step $t+1$ to be not so far from policy at time t, avoiding high changes in the policy q.

The advantage function is a measure of the extra gain that could be obtained when selecting the action a in state s ($A(s,a) = Q(s,a) - V(s)$). The autonomous agent updates the weights of its policy in the direction to maximise this advantage estimate (PPO) or the return (REINFORCE).

Policy or Value-based methods? The primary strength of policy optimisation methods is that they directly optimise the object you want, namely the policy. This tends to make them more stable and reliable. On the contrary, Q-learning methods only indirectly optimise the agent's performance, by training Q_θ to satisfy a self-consistent equation. There are many failure modes for this type of learning, so it tends to be less stable. Yet, Q-learning methods gain the advantage of being more sample efficient when they do work, because they can reuse data more effectively than policy optimisation techniques.

Moreover, policy optimisation and Q-learning are not incompatible (and under some cases, they turn out to be equivalent), and there is a range of algorithms that fall between the two extremes. Algorithms that live on this range are able to carefully arbitrate between the strengths and weaknesses of each side. Examples include *Deep Determinisitc Policy Gradient* [92] (DDPG), an algorithm that simultaneously learns a continuous deterministic policy and a Q-function by using each objects to improve the other and *Soft Actor Critic* [55] (SAC), a variant which uses stochastic policies, entropy regularisation, and a few other tricks to stabilise learning.

3.2.4 Model-based reinforcement learning

Model-based reinforcement learning (MBRL) techniques ought to work with a model of the environment to perform planning. Two principal frameworks exits : either the agent is provided the model or it has to learn it

from experience. We provide in this chapter familiarities about this class of RL methods and how it works : first how the agent can learn the world model and second how it can use the model for planning. The details about model-based algorithms (learning, planning, architectures) will be provided in Chapter V.

3.2.4.1 Learning the model

The usual scenario in reinforcement learning applications is that a ground-truth model of the environment is not available to the agent. If an agent wants to use a model to operate planning techniques, it has to learn the model purely from experience, which creates several challenges. The biggest challenge is that bias in the model can be exploited by the agent, resulting in an agent which performs well with respect to the learned model, but behaves sub-optimally (or super terribly) in the real environment. Learning the environment model is fundamentally hard, as it can take lots of time to have a good approximation and can even fail. Since several years, some researchers have tempted to propose solutions for model learning in RL environments. Recall that the environment model $\mathcal{M}$ should be an object that can predict the next state s' and the reward r when the agent takes action a in state s. We denote by $\tilde{\mathcal{M}}$ the approximated model that the agent learns from collected data. In some scenarios, the agent has to learn the dynamics of the environment $\mathcal{P}$ since the reward is hand-crafted by human knowledge or engineers and thus provided to the software agent. However, there are cases where the agent must also learn the reward. Chapter V and Chapter VI assume that the reward function is given to the agent and only the transition probabilities have to be approximated therefore we present works with this assumption.

What model to learn ? The first point asks what model the agent should learn. With environment interactions, the agent collects batch of one-step transition data $(s^t, a^t, r^{t+1}, s^{t+1})$. There exist three principal types of dynamics function [106] :

* *Forward model* that predicts the new state s^{t+1} at time $t+1$ when the agent selected action a in state s^t at time t : $(s^t, a^t) \rightarrow s^{t+1}$;
* *Backward model* that predicts the pair state-action (s^t, a) that led to the new state s^{t+1} : $(s^t, a^t) \leftarrow s^{t+1}$;
* *Inverse model* that predicts which action the agent should take to move from state s^t to state s^{t+1} : $(s^t, s^{t+1}) \leftarrow a^t$

Model-based RL has mostly focused on forward models, and these will also be the main focus of our discussion, and investigated solutions in the thesis.

How to learn the model ? Now to estimate these forward models the agent has several mathematical tools that are nicely covered in the survey [106] and that will be detailed in Section 5.1 of Chapter V :

* *Non-parametric* methods that directly store and use the data to represent the model :
  - *Replay buffers* [43, 133] : Replay buffers techniques are dealing with *deterministic models* where the collected tuple (s, a, r, s') is kept in memory (the replay buffer) for planning and updating value function or policy. Basically, it acts as a predictor for couple $(s, a) \rightarrow (r, s')$;
  - *Approximate methods* : We may also apply non-parametric and approximation methods when we want to be able to generalise information to similar states. In this context, we approximate the transition model $(s, a) \rightarrow (r, s')$ with approximated methods. For example, this can be done with neural networks, linear approximation or Gaussian processes [145] that have became a popular non-parametric approach.
* *Parametric* methods, discussed in [56] :

— *Exact tabular methods* : For a discrete MDP (or a discretised version of a continuous MDP Chapter V), a tabular method maintains a separate entry for every possible transition. For example, in a stochastic MDP (in which we need to learn a probability distribution), Sutton proposed a tabular maximum likelihood model [133] to estimate the probability of each possible transition $(s, a) \rightarrow s'$ by counting occurrences of visits in these couples (more details in Chapter V). Being able to learn this exact model of the world allows first a safe planning and exploration thus the agent can train from simulated experiences. Moreover these methods are sample efficient in the sense that they require less interactions with the environment to converge. However, they do not scale to high-dimensional problems, as the size of the required table scales exponentially in the dimensionality of S (see Chapter V and Chapter VI).

— *Approximate methods* : The dynamic model function $\mathcal{P}$ can also be approximated by $\tilde{\mathcal{P}}$, which will scale down the memory requirements and be more feasible in large scale systems. Usually, the dynamics approximator will be parametric functions such as : Linear regression [8], Dynamic Bayesian Networks [26], Neural Networks [108], etc.

As we have seen, there are a variety of challenges and issues in model learning. In the next paragraph, we will discuss how this learned (or given) model may actually be used to act and learn in the environment.

3.2.4.2 Exploiting the model - Planning

The second branch of model-based RL methods is about scenarios where the agent is given the environment model or has learned the model. The main difference here is that in the former case the agent can directly do planning based on the accurate model of the world while in the latter case the agent needs to learn the model from interactions with the environment and then do some planning. This family of reinforcement learning algorithms will be treated in Chapter IV, where it is assumed that the agent have a full knowledge of the environment model $\mathcal{M}$ and in Chapter V, Chapter VI.

Model is given The very first case is about when the agent is provided beforehand the true model of the environment $\mathcal{M}$, including dynamics and reward functions. The very first techniques are the DP algorithms to solve MDPs, described in section 3.1.2. Also more recent work such as AlphaZero [129] proposes model-based RL with Monte Carlo Tree Search (MCTS) to master the game of Go.

Model is learned - Integration of Learning and Planning The scenario where the agent has learned the model, or as we focus in this document the dynamics model $\mathcal{P}$, differs from the case where the true model is provided. Indeed, the software agent have to use its updated model carefully, depending on its confidence in the learning. This raises several issues that are presented here but which will be detailed in the Background Section A.1 of Chapter V.

∗ Which *first state* to select for the planning : It exists several options to consider :

— A straightforward approach is to *randomly* select states throughout state space. This is the case of DP methods which selects all possible states in a sweep. The major drawback of this approach is that it does not scale to high dimensional problems, since the total number of states grows exponentially in the dimensionality of the state space;

— We can also ensure that the agent only plans at *reachable states* by selecting previously *visited states* as starting points. This is the case for Dyna architectures [133] (see Chapter V);

— Selecting visited states can also be improved with a *prioritisation* of the states. For example, to obtain an ordering over the reachable states, identifying their relevancy for a next planning update might be suitable. A good example is Prioritised Replay [122] (see Chapter V) which identifies state-action pairs that likely need to be update, e.g. pairs with the highest TD-error;

— Last, the agent can do planning directly *from the current state* it observes in the environment. This puts much emphasis at finding a better solution or more information in the region where we the agent currently operates.

* *Trade-off between time for collecting data and planning time* : The challenge here is to define in the algorithms when the agent must do planning.

— Planning at each environment step : As an example, original Dyna architectures proposed by Sutton [133] makes up to a hundred planning steps after every real iteration in the environment, i.e. after each tuple (s, a, r, s');

— Planning after collecting several data : The principal example is PILCO [36] which collects data in entire episodes, and plans an entire solution after a set of new real transitions has been collected. This is also known as *Batch reinforcement learning* where the agent plans after collecting a batch of experiments.

* *How long should the planning phase last* : The planning phase is made of several iterations to browse the state and action spaces for updating value functions or policies. One plan can be seen as an 'imagined' simulation that the agent can roll from the learned model. Two principal points emerge : how large the agent should browse the state space (or start a new planned simulation), known as *breadth* and how deep the plan should be, known as *depth*. We detail below the two points and summarises it in Figure 3.3.

— *Breadth* of the plan to decide how large the agent browse the state space : Original model-based RL methods usually consider a breadth of size 1 to sample single transitions from the model and applying model-free updates, e.g. Dyna architectures. Very recent works integrate this reasoning in modern model-free techniques : e.g. with DDPG [70]. An other more flexible example is AlphaZero [129] with MCTS method that deals with adaptive breadth sizes depending on the tree arms. Last techniques are DP algorithms such as VI and PI [5] that consider a breadth of full state space size, updating the value function and the policy for all state $s \in S$ for a depth of one, with recursive equation from time t to time $t+1$. This is also the case for *MDP online* techniques [74], [27] where the agent ought to approximate reward and transition matrices with a tabular maximum likelihood model [133]. With the learned model, it can do planning by solving the underlying MDP. These two algorithms will be presented in Chapter V since they will be treated and influenced our work.

— *Depth* of the plan to decide how deep the agent plans. Original Dyna algorithms [133] considered a depth of size 1 (and breadth of size 1), i.e. the agent supplementary updates the couple (s, a) with the collected data (s, a, r, s'). In the context of Dyna, [60] shows that plans longer than depth 1 give better learning performance. Finally, PILCO [36] proposes to plan until it terminates (final state, ending condition).

Also note that the learning process can be done Online and Offline :

* *Online planning* : planning is done just before executing an action in a given state s, i.e. that planning derives the policy $q(s)$ for the given state s for example by evaluating $V(s)$ only for this state;

* *Offline planning* : solves the problem offline for all possible states (e.g. Value Iteration), therefore deriving the whole policy $q(s)$ $\forall s \in S$.

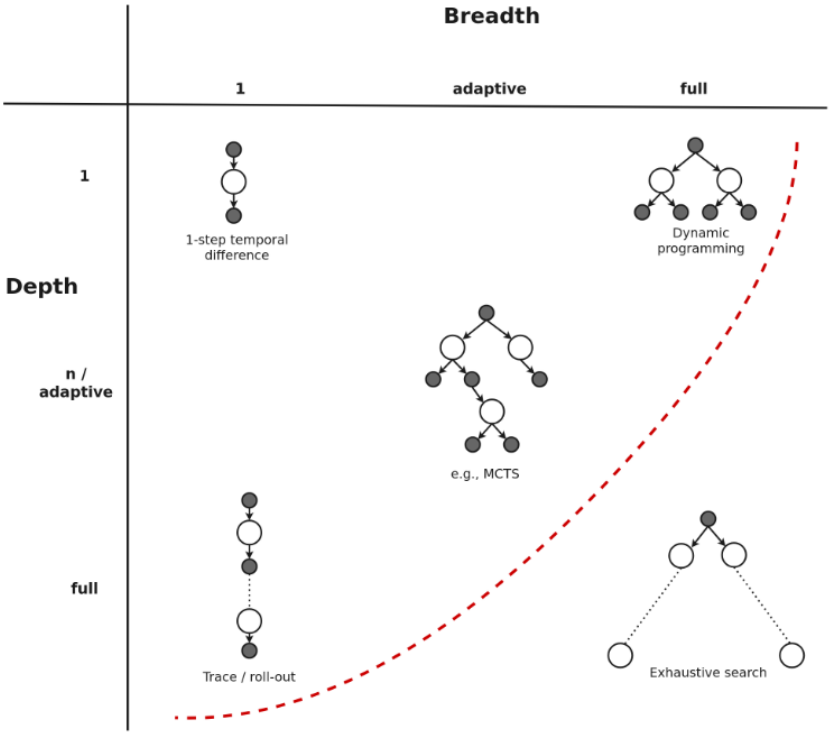


FIGURE 3.3 – Breadth and Depth for a planning iteration (Figure from [106]) - The red dot line excludes the exhaustive search unfeasible in MDP environments

Data augmentation models Finally, one other technique that can be considered as model-based-model-free RL is a setting where a world model given to the agent can be utilised to generate supplementary data, referred as *Data augmentation*. These techniques are a mixed between model-based and model-free because the learning process remains in a model-free fashion. The environment model is only used to augment the data for learning process. For example, Feinberg et al. [44] proposed *Model Based Value Expansion* (MBVE), a method that enables the use of learned dynamics models into model-free RL techniques to improve value function estimation. Moreover, this technique is used in Causal Reinforcement Learning (CRL) described in section 3.2.5.2, where the agent can generate data with the causal model of the world.

3.2.5 Reinforcement learning with structural knowledge

More recent frameworks ought to integrate relational knowledge into RL methods to improve performances. This representation is placed above the other two distinctions : model-free versus model-based, policy-based versus value-based, i.e. that this supplementary relational knowledge can fit in all RL methods. In general, it is related to model-based techniques however structural knowledge can be a benefit for model-free techniques. This section introduces the idea of factored and causal MDP frameworks that are studied in Chapter VI and Chapter VII.

3.2.5.1 Factored framework

Markov decision processes (MDPs) have proven to be popular models for decision-theoretic planning, but standard dynamic programming algorithms for solving MDPs rely on explicit, state-based specifications and computations. In other terms, usual MDP solutions represent the environment state s as a single object therefore the complexity of such approaches is often exponential in the state space size.

To alleviate the combinatorial problems associated with such methods, Boutilier [26] proposed *Factored MDP* (FMDP), a new representational and computational technique for MDPs that exploit certain types of problem structure. This framework is a feature-based representation of MDP framework. Formally, the environment state can be characterised by a finite set of random variables $s = s_1, \ldots, s_N$. With this new representation, researchers can integrate the local dependencies between the state variables s_i and can provide this knowledge to the learning agent. With this local dependencies understanding, FMDP uses Dynamic Bayesian Networks (DBNs), with decision trees or decision diagrams to represent the local conditional probability distributions and a factored reward function in a more compact form. It also allows a more compact and decomposed representation of policies and value functions.

In his work [26], Boutilier also provided first factored versions of standard dynamic programming algorithms that directly manipulate decision tree or decision diagram representations of policies and value functions. The main benefit of factored approaches is therefore a more compact representation of the model including dynamics $\mathcal{P}$ and reward $\mathcal{R}$ which reduces the number of computation in algorithms by considering local components. Therefore, FMDP solutions require smaller memory implementation for large-scale systems and yield faster convergence.

The factored framework for MDP and RL have gained attention in the last decades [83], [100], [35], [73] or [135], where authors studied different solutions, to represent, learn and plan with factored representation of the MDP. While decision tree or decision diagrams are the main representation for factored MDP framework, these solutions only deal with binary value variables which limits the scope of possible applications. On the other hand, few works such as *Factored Value Iteration* [135] ought to integrate the factored knowledge and local conditional probabilities in DP algorithms in order to treat multivariate environment variables.

This framework is at the heart of Chapter VI where detailed formalism will be given and state of the art methods will be described in the respective chapter.

3.2.5.2 Causal framework

More recently, researchers have tried to merge **Causality** with **Reinforcement Learning**. While factored framework consider associational relations between environment variables for more compact representation, the causal field ought to go further by considering causal relations, starting from the *Reichenbach's principle*, depicted in Figure 3.4 and explaining that association is different than causation.

Definition 1 (Reichenbach's principle [114]). If two random variables X and Y are statistically dependent ($X \not\perp\!\!\!\perp Y$), then there exists a third variable Z that causally influences both. (As a special case, Z may coincide with either X or Y). Furthermore, this variable Z screens X and Y from each other in the sense that given Z, they become independent, $X \perp\!\!\!\perp Y|Z$.

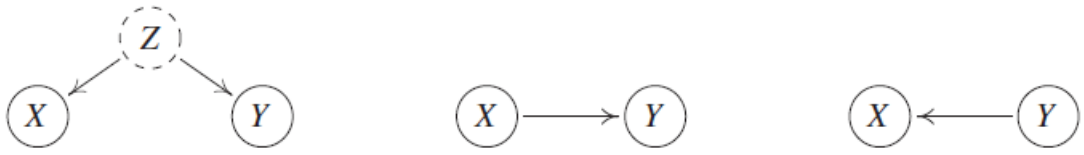


FIGURE 3.4 – Reichenbach's principle

This principle is the center of the causality field that study the causal relation between random variables. The formalism of causality owes much to Judea Pearl [3] who has built a complete mathematical formalism in the early 2000s that will be described in Chapter VII. Basically, it formulates a *Structural Causal Model* (SCM) as a

mathematical tool to illustrate how environment variables are causally related to each others. The SCM object, defined in Chapter VII, can represent all the 3 layers that Pearl uses to describe the whole causality field :

* *Observational* : The observational layer treats the system under unique observations, i.e. as the environment behaves on its own. We refer to *observational data*;
* *Interventional* : The interventional layer treats the system under interventions from an agent that can modify some environment variables. We refer to *interventional data*;
* *Counterfactual* : The counterfactual layer is about the imaginary world that an agent thinks about to evaluate counterfactuals. In other worlds, it aims to respond the following question : What would be the consequences of an other action a' in state s, knowing that I have done action a in the state s.

Very recently, Pearl stated that Reinforcement Learning was indeed the interventional layer defined by himself years ago, i.e. that the RL paradigm ought so solve dynamic optimisation problems by collecting interventional data in a system. This has lead to a new emerging field, *Causal Reinforcement Learning* (CRL), which has been built to integrate the both frameworks (see Figure 3.5) : first by helping the causality field where reinforcement learning can help to discover causal relations with environment interactions and interventional data (where the agent intervene and modify directly the value of state variables). Secondly, causal knowledge can help reinforcement learning by providing to the RL agent a deeper understanding on how environment variables are related, and more interestingly the counterfactual process for RL agents. The latter is studied in Chapter VII. The benefits that causal knowledge can bring to the reinforcement learning are several and detailed in the respective chapter. One can refer to the work of Gershman [47] that reviewed the diverse roles that causal knowledge can play in model-based and model-free RL.

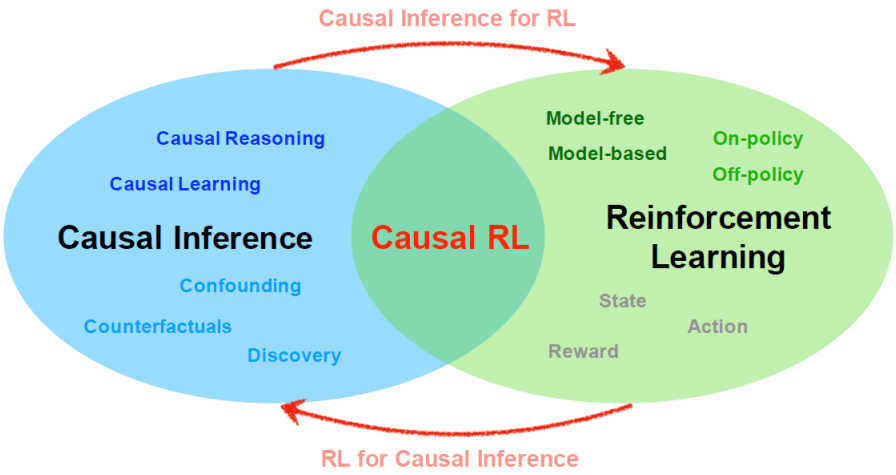


FIGURE 3.5 – Causal Reinforcement Learning : How can each field help the other (Figure from [96])

More formalism and state of the art algorithms will be described in the Chapter VII. Moreover, this respective chapter takes up all the reinforcement learning distinctions seen in this chapter and proposes a unified vision of all approaches presented in this chapter.

3.2.6 Summary of RL algorithms

We display in Figure 3.6 the global decomposition of RL algorithms with the two principal orthogonal divergence we presented : model-free versus model-based, policy-based versus value-based. Last, the figure decompose

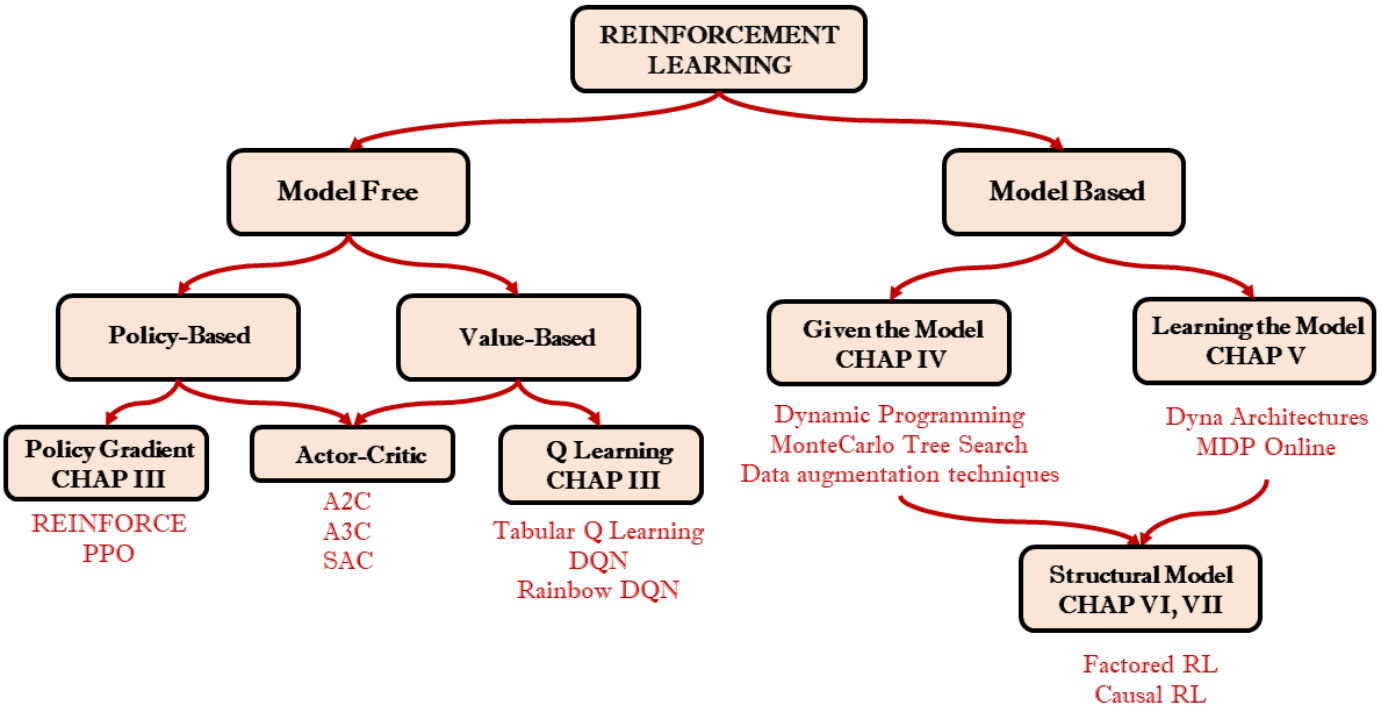


FIGURE 3.6 – Taxonomy of reinforcement learning methods and their application in the thesis

the model-based class into two sub-families : the first one aims to learn the model by interaction with the environment and use it for planning while the second class assumes that the model is given to the agent (e.g. see MDP methods). We provide the application and study of class of RL algorithms in the respective chapters.

3.2.7 Comparison criteria in reinforcement learning

The last major component in the reinforcement learning field is how researchers can compare efficiently algorithms [34, 106]. In order to identify benefits of designed RL algorithms, researchers need to discuss performance criteria, and establish terminology about the two types of exploration in model-based RL. Usually, there are two main evaluation criteria for RL techniques :

* *Cumulative or average reward* expressing the quality of the solution.
* *Time complexity* expressing the amount of time needed to arrive at the solution (calculations complexity, number of trials in real or simulated environment, sample efficiency)

Researchers usually report learning curves, which show optimality (cumulative or average return) on the y-axis and one of the above time complexity measures on the x-axis over the learning process. Solutions can also been evaluated offline after the learning process in the same manner by looking at the obtained return following the policy. As we will see, (structural) model-based RL may actually be used to improve both measures. More details about comparison between RL algorithms will be given in Chapter V and will be used in the following chapters.

3.3 Model-free or model-based RL ?

This section discusses the use of model-free or model-based methods in practice. We first briefly cover two industrial networking applications that were performed during my Cifre thesis where we applied model-free RL techniques and finally discuss about which RL class to use in practice by giving pros and cons.

3.3.1 Industrial networking applications

3.3.1.1 Grid of Beams

Wireless network coverage is defined as the extent of the area to which the wireless signals are transmitted. The coverage is beam-based in $5G$ networks, not cell based. There is no cell-level reference channel from where the coverage of the cell could be measured. Instead, each cell has one or multiple Synchronisation Signal Block Beam (SSB) beams. In legacy systems, SSB beams are static, or semi-static, always pointing to the same direction. They form a grid of beams covering the whole cell area. The UE (user equipment) searches for and measure the beams, maintaining a set of candidate beams. The candidate set of beams may contain beams from multiple cells. The metrics measured are RSRP (Reference Signal Received Power), RSRQ (Reference Signal Received Quality), RSSI (Received Signal Strength Indicator) for each beam, that tells how good the received signal is for the UEs. Therefore there is a need to optimise the set of candidate beams that will maximise the coverage for the UEs. We performed a feasibility study of Deep RL techniques (DQN) in this networking problem that can be seen as a combinatorial optimisation problem. The main finding was that the DQN was outperformed by other classical optimisation techniques such as submodular optimisation and local search algorithms. The major issue of the DQN was its convergence time. One of the conclusions of this work is that if we have knowledge of the model (dynamics, reward function) then we must use it for offline planning and optimisation algorithms.

3.3.1.2 PDCP Data Split

We also applied Deep RL techniques to an other networking use-case : PDCP (Packet Data Convergence Protocol) Data Split in Dual Connectivity (DC) scenario. The PDCP layer handles transfer of user data, header compression, sequence numbering, duplication detection, packet duplication, etc. For the sake of describing the DC solution, we consider an LTE scenario and adopt the LTE terminology. With DC, a UE is simultaneously connected to two different base stations : a master eNB (MeNB) and a secondary eNB (SeNB). The MeNB and the SeNB are connected via a non-ideal backhaul and operate on different carrier frequencies.

We were interested in optimising how to split the Packet Data Units (PDUs) sent by the UEs in the network, i.e. in which path the PDU should be forwarded to optimise latency and avoid overall congestion. In this context, we do not have a perfect modelling of the environment (dynamics and reward which is difficult to measure in real time). Moreover, the action space for a learning agent is relatively simple since it only has very few choices (where to send incoming PDUs). For this scenario, Deep RL methods were able to improve existing SoTA techniques implemented in practice, demonstrating real gain of applying model-free RL techniques.

3.3.2 Thesis position and overview of model-free versus model-based techniques

First of all, let us remind pros and cons for both model-free and model-based RL methods.

— *Model-free* :
  — Pros : Computationally less complex and needs no accurate representation of the environment in order to be effective.
  — Cons : Actual experiences need to be gathered in order for training, which makes exploration more dangerous. Cannot carry an explicit plan of how environmental dynamics affects the system, especially in response to an action previously taken.

— *Model-based* :
  — Pros : Safe to plan exploration and can train from simulated experiences. Moreover they are sample efficient in the sense that they require less interactions with the environment to converge.

— Cons : The learning agent is only as good as the model learnt. Sometimes this becomes a bottleneck, as the model becomes surprisingly tricky to learn. Finally, model-based methods are computationally more complex than model-free methods.

In general, most of the reinforcement learning applications appear to be model-free techniques whether in the literature or in industrial applications, mainly because of its ease of implementation and consideration. However, convergence problems are often mentioned when applying model-free techniques and this is simply because the autonomous agent should explore infinitely many times all the state-action pairs to obtain a good evaluation of the value function and the policy. Nevertheless, we have seen in practice but also for general use-cases that if we have some information about the environment model, researchers should investigate this knowledge to build or assess optimisation methods. The goal of this thesis is to evaluate and propose model-based reinforcement learning techniques to accelerate the convergence and provide the agent with supplementary knowledge, in a Cloud resource allocation use-case. It is believed that providing the model to the autonomous agent seems to be a major benefit that should be integrated in its knowledge and that researches on model-based RL industrial applications should be investigated. Chapter IV treats the scenario where the agent is given the model $\mathcal{M}$, and can be also solutions to employ after the learning of the environment model. Chapter V treats the scenario where $\mathcal{M}$, specially the dynamics $\mathcal{P}$ is unknown but the agent tries to approximate the dynamics model to perform planning. Chapter VI and Chapter VII are model-based reinforcement learning proposal with factored and causal knowledge of the environment. The latter can also treat complex environments with confounders and unobserved variables which can be a huge issue in model-free techniques that will learn biased policy or value functions.

PART II

MODEL-BASED APPROACHES FOR CLOUD RESOURCE ALLOCATION

CHAPTER 4

MARKOV DECISION PROCESS OR HEURISTICS : HOW TO COMPUTE AUTO-SCALING HYSTERESIS POLICIES IN QUEUING SYSTEMS WITH KNOWN MODEL

The contributions of this chapter have been published in [139, 141] respectively :

* Thomas Tournaire, Hind Castel-Taleb, Emmanuel Hyon. Generating optimal thresholds in a hysteresis queue : a cloud application. In *27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 19)*, 2019.
* Thomas Tournaire, Hind Castel-Taleb, Emmanuel Hyon. Optimal control policies for resource allocation in the Cloud : comparison between Markov decision process and heuristic approaches. In *CoRR abs/2104.14879 (2021)*, 2021.

4.1 Introduction

This chapter considers that the agent is provided the true model of the environment $\mathcal{M}$ therefore its scope falls into the class *Given the model* of Figure 3.6. We consider a cost-aware approach and propose a global expected cost considering different costs : associated with the performance requirements defined in a Service Level Agreement (SLA) and associated with energy consumption. We want to compute the optimal policy of activation and deactivation of the servers according to the queue occupation. In this chapter we study models in which the structure of the policy is of hysteresis form. However, assuming the policy has an hysteresis form is not sufficient to fully characterise the optimal policy. We have to determine the concrete values of thresholds. For this aim, there exist two modelling approaches which correspond to two major streams in the literature. The first one expresses the problem into a model of deterministic optimisation problem. This is done by using the average value of the costs computed by stationary distribution of a CTMC (Continuous Time Markov Chains). Such an approach is often used in the field of inventory management for base stock policies [146] or reliable servers [149]. The second one expresses the problem under a MDP (Markov Decision Process) model with structured polices and computes the optimal one. But, although they are the most widely used for threshold computation, the efficiency of these two approaches are not assessed very carefully for the computation of single thresholds by level [6]. Especially, this had never been done for hysteresis policies. Few works focused on the optimality of hysteresis policy for cloud models [151] but, up to our knowledge, none on the threshold computation. Determining which is the best promising approach and which is the best model to choose in a practical use is thus an important point which is addressed

here.

We also validate our approach with a real model. The approach is difficult since, up to our knowledge, it does not exist, in the literature, an unified model including both energy, quality of service, and real traffic parameters. We propose to build a global cost taking into account both real energy consumption, real financial costs of virtual machines and Service Level Agreement (SLA) simultaneously from separate works, and we give real environment values to our models. We think that the model presented is sufficiently generic to represent a large class of problems and is enough to give a relevant meaning to all parameters. With our algorithms and this model cloud owners can generate meaningful optimised costs in real cases.

The contributions of this chapter are the following :

* Multi-server queuing model for a cloud IaaS infrastructure with a single physical node hosting several virtual resources;
* Re-formalism and building a correspondence between literature hysteresis definitions;
* Comparison of many local search heuristics and meta-heuristic based on the computation of the stationary distribution of the Markov chain to find optimal hysteresis rules and proposition of an aggregation technique on the Markov chain to accelerate the computation of the mean cost;
* Theoretical and numerical comparison with Markov decision process algorithms where we provide a theoretical study of the multichain properties of the Markov decision process which shows that some usual algorithms solving MDP models can fail. We nevertheless provide an algorithm that solves dynamic control model with convergence guarantee. We also design dynamic control algorithms integrating hysteresis policies and we show that they are faster while ensuring the optimality of the solution;
* We perform a large set of numerical experiments and exhibit that dynamic control approach strongly outperforms the other one for optimality and running time criteria. We make theoretical analysis and give some insights explaining why the deterministic optimisation problem is sub-optimal;
* Exploration of numerical experiments to check if the optimal policy is of hysteresis type;
* We develop and analyse a financial cost model. It takes into account prices of VM instantiations from cloud providers as well as energy consumption of VMs. A presentation of minimised costs for a problem based on this concrete cloud model is done. We noticed that optimal thresholds are generated in just few seconds even for large systems which could have a significant impact for a cloud operator.

Reminders about basics formalism for queuing systems and computation of stationary distribution in associated Markov chains are provided in the appendix A.

4.2 Cloud Model

4.2.1 Cloud use-case

Cloud computing includes three major delivery models as described in Chapter II : Software-as-a-Service (SaaS) in which the consumer is able to use an application to meet specific needs, Platform-as-a-Service (PaaS) which provides the consumer with an hosting environment for application development, and Infrastructure-as-a-Service (IaaS) in which the consumer has a greater access to computing resources including processing power, storage, and networking components (see Fig. 4.1).

In this section, we present the cloud system, denoted *Cloud Model*. This is a continuous time system modelled by a continuous time controlled multi-server queueing model, where arrivals of requests are processed by servers. The servers in the queueing system represent logical units in the cloud. Since we consider IaaS clouds (Infrastructure as a Service) which provide computing resources as a service, the logical units are either virtual machines or

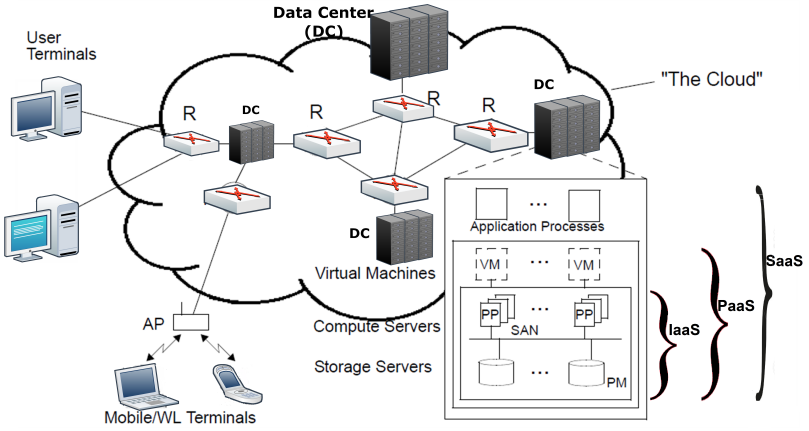


FIGURE 4.1 – Representation of a Cloud Architecture

docker components. We propose a mathematical analysis of the queueing model in order to derive performance as well as energy consumption measures.

4.2.2 Controlled multi-server queue

The Cloud model is a physical node hosting several virtual resources with a multi-server queuing model depicted in Figure 4.2 and described as follows :

1. Arrivals of requests follow a Poisson process of rate λ, and service times of all VMs are independent of arrivals and independent of each other. Moreover, they are i.i.d. and follow an exponential distribution with an identical rate μ;
2. Requests enter in the queue, which has a finite capacity B, and are served by one of the K servers. The service discipline is supposed FIFO (First In First Out).

A customer is treated by a server as soon as this server becomes idle and the server is active. Servers can be turned on and off by the controller. When the server is turned on it becomes active while it becomes inactive when it is turned off.

We define $\mathcal{S}$ the state space, where $\mathcal{S} = \{0, 1, \ldots, B\} \times \{1, \ldots, K\}$. Any state $x \in \mathcal{S}$ is such that $x = (m, k)$ where m represents the number of requests in the system, and k is the number of operational servers (or active servers, this number can also be seen as a service level). We define $\mathcal{A} = \{0, \ldots, K\}$, be the set of actions, where action a denotes the number of servers to be activated. With this system, are associated two kinds of costs that a cloud provider encounters :

1. Costs corresponding to the performance of the system, for the control of the service quality defined in the SLA : as costs (C_H) per unit of time for holding requests in the queue or instantaneous costs (C_R) for losses of requests.
2. Costs corresponding to the use of resources (operational and energy consumption) : as costs for using a VM per unit of time (C_S) and instantaneous costs for activating/deactivating (C_A and C_D).

We define for the system a global cost containing both performance and energy consumption costs. We have the following objective function we want to minimise :

$$\bar{C} = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}\left\{ \int_0^T C_t\left(X(t), A(t)\right) dt \right\} \tag{4.1}$$

where $A(t)$ is the action taken at time t (it is possible that nothing was performed) and $C_t(X(t), A(t))$ is the cost that is charged over time when the state is $X(t)$ and action $A(t)$ is performed. This problem can be solved in two different ways based on two different models : using a Continuous Time Markov Chain to solve deterministic optimisation problem (see section 4.3.2) or using Semi Markov Decision Processes (see section 4.6).

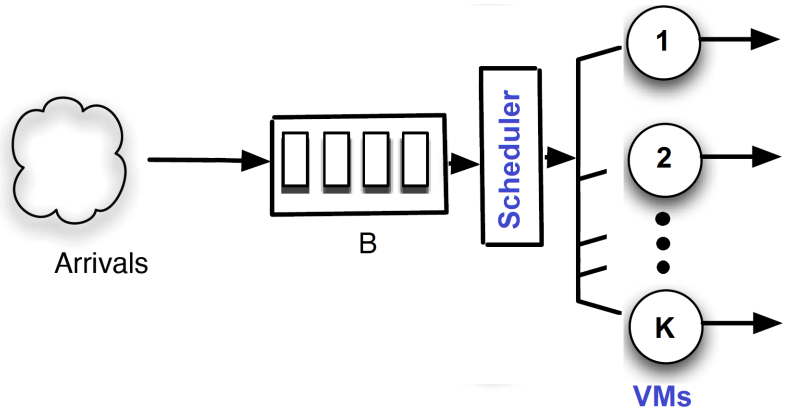


FIGURE 4.2 – Multi-server queuing system

4.3 Hysteresis policies and hysteresis queuing model

4.3.1 Hysteresis policies

A decision rule is a mapping from some information set to some action. A policy is a sequence of decision rules $\eta = (q_0, q_1, q_2, \ldots)$. The most general set of policies is that of history-dependent randomised policies, but the classical results on average infinite-horizon, time-homogeneous Markovian optimal control [5] allow us to focus on stationary Markov Deterministic Policies. Such policies are characterised by a single, deterministic decision rule which maps the current state to an action. We thus consider the mapping q from $\mathcal{S}$ to $\mathcal{A}$ such that $q(x) = a$.

All along the chapter we restrict our attention with a special form of policies : the hysteresis policies. Nevertheless, there are few homogeneities between definitions of hysteresis policies in the literature and we provide here a correspondence between them. Indeed, it can refer to policies defined with double thresholds (especially when $K = 2$), or to a restrictive definition when Markov chains are used. We follow the works of [58, 78] to present an unified treatment of hysteresis.

Hysteresis policies with multiple activations We assume in this part that the decision rule is a mapping from the state to a number of active servers $q(m, k) = k_1$ with $k_1 \in [1, \ldots, K]$. Several servers can be activated or deactivated to pass from k to k_1 active servers.

Definition 2 (Double threshold policies [58, 78]). We call *double threshold* policy a stationary policy such that the decision rule $q(m, k)$ is increasing in both of its arguments and is defined by a sequence of thresholds such that for any k and k_1 in $[1, K]$ we have :

$$q(m, k) = k_1 \text{ when } \quad \ell_{k_1}(k) \leq m < \ell_{k_1+1}(k)\,,$$

where $\ell_{k_1}(k) = \min\{m : q(m, k) \geq k_1\}$. This minimum is ∞ if the set is empty. For all k, we also fix $\ell_{K+1}(k) = \infty$ and $\ell_1(k) = 0$ (since at least one server must be active).

A monotone hysteresis policy is a special case of double threshold policy.

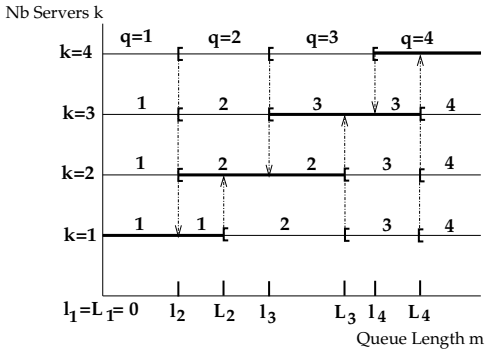


FIGURE 4.3 – An isotone hysteresis policy.

Definition 3 (Monotone Hysteresis polices [58]). A policy is a monotone hysteresis policy if it is a double threshold policy and moreover if there exist two sequences of integers l_k and L_k such that

$$l_k = \ell_k(K) = \ell_k(K-1) = \ldots = \ell_k(k)$$
$$L_k = \ell_k(1) = \ell_k(2) = \ldots = \ell_k(k-1)\,,$$

with $l_k \leq L_k$ for $k = 1, \ldots, K+1$; $l_k \leq L_{k+1}$ for $k = 1, \ldots, K$; $l_1 = L_1 = 0, l_{K+1} = L_{K+1} = \infty$.

And if, for all $(m,k) \in \mathcal{S}$,

$$q(m,k) = \begin{cases} q(m,k-1) & \text{if } m < l_k \text{ and } k > 1 \\ k & \text{if } l_k \leq m < L_{k+1} \\ q(m,k+1) & \text{if } m \geq L_{k+1} \text{ and } k < m \end{cases}.$$

The thresholds l_k can be seen as the queue levels at which some servers should be deactivated and the L_{k+1} are the analogous activation points. Roughly speaking, the difference between a double threshold and a monotone hysteresis policy lies in the fact that some thresholds toward a level are identical in hysteresis.

Definition 4 (Isotone Hysteresis polices [58]).] A policy is an isotone policy if it is a monotone hysteresis policy and if $0 = l_1 \leq l_2 \leq \ldots \leq l_{K+1} = \infty$ and $0 = L_1 \leq L_2 \leq \ldots \leq L_{K+1} = \infty$.

Example 1. *In Figure 4.3, we represent an isotone policy. The number indicates the number of servers that should be activated in each state. The bold line means that the number of activated servers is the same than the decision and then no activation or deactivation have to be performed.*

Proposition 1 (Optimality of monotone hysteresis policies [134]). *In the multi-server queueing model presented in Section 4.2 and for which there is activation/deactivation costs, working costs and holding costs, Szarkowicz and al. showed that monotone hysteresis policies are optimal policies.*

However, the presence of a rejection cost, as we assume here, is not considered in the assumptions of [134] and there is no proof about the optimality of hysteresis policies for the model studied here.

Remark. There is an alternate way to express the policy by giving the number of servers to activate or deactivate instead of giving directly the number of servers that should be active.

Hysteresis policies with single VM activation We focus now on models in which we can only operate one machine at a time. Now the decision rule indicates an activation or deactivation. Therefore, the action space is now $\mathcal{A} = \{-1, 0, 1\}$. For a state (m, k), when the action $q \in \mathcal{A}$ is -1 then we deactivate a server, when the action is 0 then the number of active servers remains identical, when the action is 1 then we activate a server.

It could be noticed that, for this kind of model, double threshold policies and hysteresis policies coincide. Indeed, the decision now relates to activation (resp. deactivation) and no longer to the number of servers to activate (resp. deactivate). There exist only two thresholds by level k : L_{k+1} to go to level $k+1$, and l_k to go to level $k-1$. There are no other levels that can be reached from k. For example, in Figure 4.3 all decisions smaller than the level are replaced by -1 while all decisions larger than the level are replaced by 1. Definition 4 remains unchanged, but Definition 3 should be rephrased in :

Definition 5 (Monotone hysteresis policy). A policy is a monotone hysteresis policy if it is a stationary policy such that the decision rule $q(m, k)$ is increasing in m and decreasing in k and if is defined by two sequences of thresholds l_k and L_k such that for all $(m, k) \in \mathcal{S}$:

$$q(m,k) = \begin{cases} -1 & \text{if } m < l_k \text{ and } k > 1 \\ 0 & \text{if } l_k \leq m < L_{k+1} \\ 1 & \text{if } m \geq L_{k+1} \text{ and } k < m \end{cases},$$

with $l_k \leq L_k$ for $k = 1, \ldots, K+1$; $l_k \leq L_{k+1}$ for $k = 1, \ldots, K$ and $l_1 = L_1 = 0$; $l_{K+1} = L_{K+1} = \infty$.

Hysteresis policies and Markov chain As presented in [137], there exists a different model of multi-server queue with hysteresis policy which received a lot of attention (see Ibe and Keilson [63] or Lui and Golubchik [98] and references therein). This model is still a multi-server queueing system but is no longer a controlled model : the transitions between levels are also governed by sequences of prefixed thresholds this is why it is called an hysteresis model. It is built to be easily represented by a Markov chain. The differences between the controlled model and this Markov chain model are detailed in Section 4.6.3.

Remark. For convenience, deactivation and activation thresholds will be denoted respectively by R and F in the Markov chain model, while they will be denoted l and L in the MDP model.

Definition 6 (Hysteresis policy [98]). A K-server threshold-based queueing system with hysteresis is defined by a sequence $F = [F_1, F_2, \ldots, F_{K-1}]$ of activation thresholds and a sequence $[R_1, R_2, \ldots, R_{K-1}]$ of deactivation thresholds. For $1 \leq k < K$, the threshold F_k makes the system goes from level k to level $k+1$ when a customer arrives with k active servers and F_k customers in the system. Conversely, the threshold R_k makes the system goes from level $k+1$ to level k when a customer leaves with $k+1$ active servers and $R_k + 1$ customers in the system.

Furthermore, we assume that $F_1 < F_2 < \ldots < F_{K-1} \leq K$, $1 \leq R_1 < R_2 < \ldots < R_{K-1}$, and $R_k < F_k$, $\forall\, 1 \leq k \leq K-1$. We denote the vector that gathers the two threshold vectors F and R by $[F, R]$.

Remark. It can be noticed that, no server can remain idle all the time here since the threshold values are bounded, whereas in Definition 5 when a threshold is infinite the server remains inactive. Hence the hysteresis policy presented in [63] can be seen as a restricted version of isotone policies given in 5. Furthermore, the inequalities being strict in [98], the hysteresis of Definition 6 is a very specific case of hysteresis of Definition 5 that we call strictly isotone.

4.3.2 Hysteresis queuing model and the associated Markov chain

This section is devoted to the study of the policies defined in Definition 6 and their aggregation properties. We consider the controlled multi-server queue model defined in Section 4.2.2 for which we integrate the hysteresis

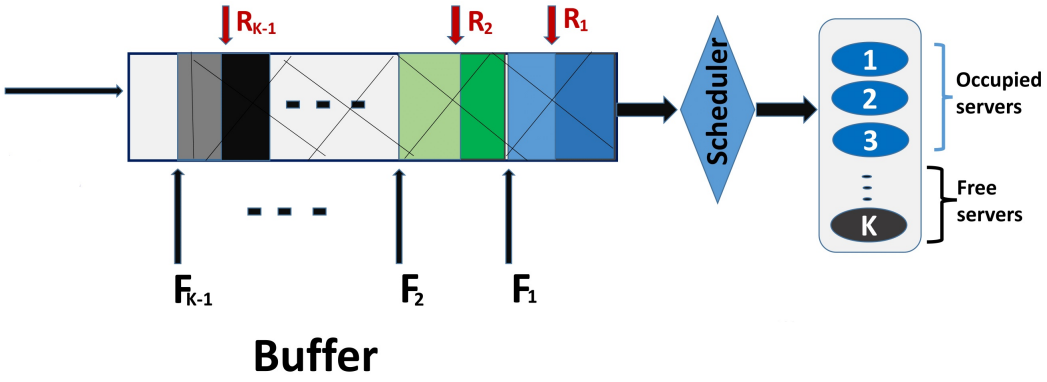


FIGURE 4.4 – The hysteresis queuing model

queuing properties. The model is characterised by a vector of activation thresholds $F = [F_1, F_2, ..., F_{K-1}]$, where F_k for $1 \leq k < K$ makes the system goes from level k to level $k+1$ when the number of customers in the queue is equal to F_k. The model is also characterised by a vector of deactivation thresholds $R = [R_1, R_2, ..., R_{K-1}]$, where R_k for $1 \leq k < K$ makes the system goes from level $k+1$ to level k when the number of customers in the queue is equal to $R_k + 1$. We also suppose that $F_1 < F_2 < ... < F_{K-1} < B$, and $R_1 < R_2 < ... < R_{K-1}$ and from the hysteresis hypothesis that $R_k < F_k, \forall k, 1 \leq k < K$. The vector that merges the activation threshold vector F and the deactivation threshold vector R is denoted by $[F, R]$. The model is depicted in Figure 4.4.

In the hysteresis model of Definition 6 thresholds are fixed before the system works. Once the thresholds $[F, R]$ are fixed, the underlying model is described by a Continuous-Time Markov Chain (CTMC), denoted $\{X(t)\}_{t\geq 0}$. Each state $(m,\ k)$ is such that m is the number of requests in the system and k is the number of active servers. The state space is denoted by $\mathcal{X}$ with $\mathcal{X} \subset \mathcal{S}$ since we do not consider non reachable states of S. Thus, the state space is given by :

$$\mathcal{X} = \{(m,\ k)\,|\,0 \leq m \leq F_1,\ \text{if}\ k = 1\,, R_{k-1}{+}1 \leq m \leq F_k, \text{if}\ 1 < k < K\,, R_{K-1}{+}1 \leq m \leq B, \text{if}\ k = K\}. \tag{4.2}$$

The transitions between states are described by :

$$\begin{aligned}(m,k) &\rightarrow (\min\{B, m+1\}, k),\ \text{with rate}\ \lambda\,,\ \ \text{if}\ m < F_k\,;\\ &\rightarrow (\min\{B, m+1\}, \min\{K,\ k+1\}), \text{with rate}\ \lambda\,,\ \ \text{if}\ m = F_k\,;\\ &\rightarrow (\max\{0, m-1\}, k),\ \text{with rate}\ \mu{\cdot}\min\{m, k\},\ \text{if}\ m > R_{k-1}+1;\\ &\rightarrow (\max\{0, m-1\}, \max\{1, k-1\})\ \text{with rate}\ \mu{\cdot}\min\{k, m\}\,,\ \text{if}\ m = R_{k-1}+1\,.\end{aligned}$$

In Figure 4.5, we give an example of these transitions for a maximum number of requests in the system equal to B and a number of levels K equal to three.

4.3.3 Cloud provider global cost

We define the mean cost that a cloud provider encounters in the cloud system. It takes into account the performance and the use of resources. It is computed knowing the stationary distribution π of the Markov Chain $\{X(t)\}_{t\geq 0}$, for a given vector of thresholds. The global cost defined in Equation (4.1) can be rewritten for fixed

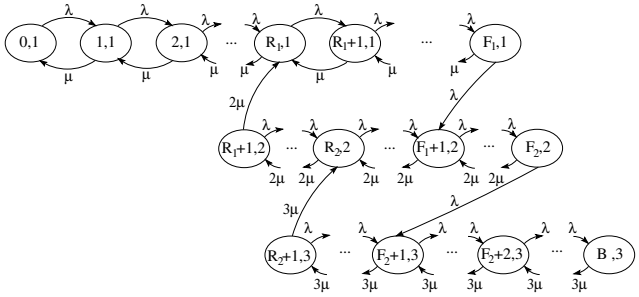


FIGURE 4.5 – Example with K=3 VMs and B maximum requests in the system

$[F, R]$. We have :

$$\overline{C}^{\pi}_{[F,R]} = \sum_{k=1}^{K} \sum_{m=R_{k-1}+1}^{F_k} \pi(m,k) \cdot \overline{C}(m,k)\,, \tag{4.3}$$

where $\pi(m,k)$ represents the stationary probability in state (m,k). For a given state of the environment (number of requests, number of activated VMs), the cloud provider pays a cost :

$$\overline{C}(m,k) = C_H \cdot m + C_S \cdot k + C_A \cdot \lambda \cdot \mathbb{1}_{m=F_k,\, k<K} + C_D \cdot \mu \cdot \min\{m,k\} \cdot \mathbb{1}_{m=R_{k-1}+1,\, 2\leq k\leq K} + C_R \cdot \lambda \cdot \mathbb{1}_{m=B,\, k=K}\,. \tag{4.4}$$

4.4 Aggregation methods to compute the global cost

One of the contributions of this chapter is to propose an aggregation technique to accelerate the calculation of the cost. Indeed, solving the Markov chain quickly becomes complex when the number of levels K increases : although it exists closed forms of the stationary distribution, their computations suffer from numerical stability problems. To overcome this issue, we propose to apply the SCA (Stochastic Complement Analysis) method [98], based on a decomposition and an aggregation of the Markov chain. This approach allows a simplified computation of the exact stationary distribution π. The aim is to separate the K levels of the considered Markov chain into K independent sub-chains, called micro-chains and then to solve each of them independently. We then build the aggregated Markov chain which has K states such that each state represents an operating level. This chain is called macro-chain.

4.4.1 Description of micro-chains and macro-chain

4.4.1.1 Micro-chains

Each micro-chain of a level k, such that $1 \leq k \leq K$, is built by keeping the transitions, of the initial Markov Chain, inside the same level k. On the other hand, the transitions between states of different levels are cut and replaced by new transitions inside the micro-chains from a leaving state to an entering state of the initial chain. For example, the transitions inside the states of level 2 with rates λ and 2μ in the Fig. 4.5 are kept, while the links between level 2 and level 3, and level 2 and level 1 are deleted, as it can be seen in Fig. 4.6. Then, the cut transitions between levels 2 and 3 in the initial Markov chain of Fig. 4.5 are replaced by a transition between $(F_2, 2)$ and $(R_2, 2)$, with rate λ, and a transition between $(R_2 + 1, 3)$ and $(F_2 + 1, 3)$ with rate 3μ (see Fig. 4.6). The approach is similar for the other transitions between other levels and for any number of levels.

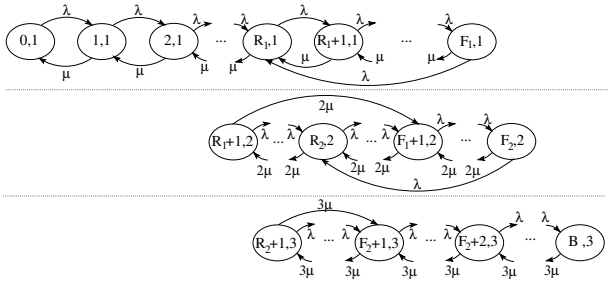


FIGURE 4.6 – Example of micro-chains from Markov Chain of Fig.4.5

4.4.1.2 Macro-chains

Concerning the macro-chain, it is described by a Markov chain with K states, where each state k (with $1 \leq k \leq K$) represents the level. The macro-chain is actually a birth-death process for which the resolution is usual with the following transition values :

$$\begin{cases} \lambda_k &= \lambda \cdot \pi_k(F_k) & \forall\, k &= 1, \ldots, K-1 \\ \mu_k &= k \cdot \mu \cdot \pi_k(R_{k-1}+1) & \forall\, k &= 2, \ldots, K \end{cases} .$$

We denote by Π the stationary probability distribution of the macro-chain and display the macro-chain in Figure 4.7.

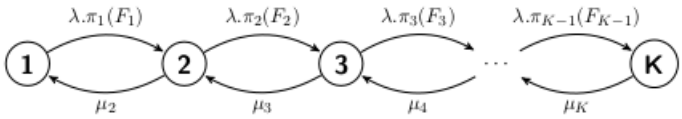


FIGURE 4.7 – The Macro chain

We denote by π_k the stationary probability distribution of the micro-chain of level k (with $1 \leq k \leq K$) and we denote by π the stationary distribution of the initial Markov chain. We obtain the distribution of the initial chain with the following equation :

$$\forall\, (m,k) \in \mathcal{X},\ \pi(m,k) \ = \ \Pi(k) \cdot \pi_k(m)\,. \tag{4.5}$$

The reader can see [98] for further details on the method. Each micro-chain will be solved independently, then the birth-death process and finally the model chain.

4.4.1.3 Stationary distribution computation

Since, as pointed out in [71], the use of closed formulas suffers from numerical instability, the solving of Markov chains (micro and macro) is done numerically using the power method or GTH algorithm implemented in the *marmoteCore* software [66] (see section A.1). Each micro-chain is solved separately, next the macro-chain, and lastly, with Eq. 1, the initial Markov chain. The distributions obtained with the aggregation process are compared with those obtained by direct computations on the whole chain. The difference is always smaller than 10^{-8} which is the precision we choose for our computations.

4.4.2 Computation of the mean cost on the aggregated chain

We use the aggregation method described in section 4.4, to aggregate the global cost $\overline{C}^{\pi}_{[F,R]}$, from Eq. (4.3), as a function of costs per level computed on the micro-chains.

Theorem 1 (Aggregated Cost Function). *The mean cost $\overline{C}^{\pi}_{[F,R]}$ of the system with fixed thresholds $[F, R]$ can be written under the form :*

$$\overline{C}^{\pi}_{[F,R]} = \sum_{k=1}^{K} \Pi(k) \cdot \overline{C}(k)\,,$$

where $\overline{C}(k)$ represents the mean cost of level k with :

$$\overline{C}(k) = \sum_{m=R_{k-1}+1}^{F_k} \pi_k(m) \cdot \overline{C}(m,k)\,.$$

Proof. We use Eq. (4.3) :

$$\overline{C}^{\pi}_{[F,R]} = \sum_{k=1}^{K} \sum_{m=R_{k-1}+1}^{F_k} \pi(m,k) \cdot \overline{C}(m,k)\,. \tag{4.6}$$

However, we know from the aggregation method and from the chain decomposition in micro-chains and macro-chain, that $\pi(m,k) = \pi_k(m) \cdot \Pi(k)$ (from Eq. (1)).

This implies :

$$\overline{C}^{\pi}_{[F,R]} = \sum_{k=1}^{K} \sum_{m=R_{k-1}+1}^{F_k} \pi_k(m) \cdot \Pi(k) \cdot \overline{C}(m,k)\,.$$

Removing $\Pi(k)$ from the second sum (since it does not depend on m), we get :

$$\overline{C}^{\pi}_{[F,R]} = \sum_{k=1}^{K} \Pi(k) \cdot \underbrace{\sum_{m=R_{k-1}+1}^{F_k} \pi_k(m) \cdot \overline{C}(m,k)}_{\overline{C}(k) \text{ mean cost per level}}\,.$$

This yields the average aggregated costs per level weighted by the macro-chain stationary distribution :

$$\overline{C}^{\pi}_{[F,R]} = \sum_{k=1}^{K} \Pi(k) \cdot \overline{C}(k)\,.$$

□

The relevance of Theorem 1 is to propose an efficient approach for the mean cost computation. Instead of computing the cost on a multidimensional Markov chain, we compute it from several one dimensional Markov chains. Next in section 4.5.2, we will take advantage of this aggregated expression of the cost coupled with a property on threshold changes, to improve the speed of the mean cost computation.

Corollary 2. *Let $[F, R]$ be a fixed vector of thresholds. It is assumed that the micro-chains and the costs per level associated with $[F, R]$ are already computed. The modification of a threshold F_k or R_k in $[F, R]$ only impacts the levels k and $k+1$ in the Markov chain. Thus, the computation of the new average cost only requires a new computation of π_k, π_{k+1}, and Π as well as $\overline{C}(k)$ and $\overline{C}(k+1)$.*

The intuition about the corollary 2 can be guessed from the transition graph in Figure 4.5. When modifying F_2, it only impacts the distributions π_2 and π_3. We thus have to compute new $\overline{C}(2)$, $\overline{C}(3)$ and Π, while the last expected cost $\overline{C}(1)$ remains unchanged. This is shown in the following proof.

Proof. Recall that we define the load of the system by $\rho = \lambda/\mu$. Using the balance equations in [98], we exhibit the impact generated by the modification of activation and deactivation thresholds on the stationary probability distributions of the micro-chains first and then the macro-chain.

Impact on the micro-chains : Owing to $m \in [R_{k-1}+1, F_k]$ for all $k \in \{1, \ldots, K\}$, we can express the stationary probability of each state in the level k in the following way :

$$\pi_k(m) = \pi_k(R_{k-1}+1) \cdot \gamma_m^k \,,$$

where (from [98]) :

$$\gamma_m^k = \begin{cases} \sum_{j=0}^{m-R_{k-1}-1} (\frac{\rho}{k})^j \,, \\ \qquad\qquad \text{if } R_{k-1}+1 \leq m \leq R_k \,; \\ \sum_{j=0}^{m-R_{k-1}-1} (\frac{\rho}{k})^j - \gamma_{F_k}^k \cdot \sum_{j=1}^{m-R_k} (\frac{\rho}{k})^j \,, \\ \qquad\qquad \text{if } R_k+1 \leq m \leq F_{k-1}+1 \,; \\ \sum_{j=m-F_{k-1}-1}^{m-R_{k-1}-1} (\frac{\rho}{k})^j - \gamma_{F_k}^k \cdot \sum_{j=1}^{m-R_k} (\frac{\rho}{k})^j \,, \\ \qquad\qquad \text{if } F_{k-1}+2 \leq m \leq F_k - 1 \,; \\ \frac{\rho}{k+\rho(\frac{\rho}{k})^{F_k-R_k}} \left[(\frac{\rho}{k})^{F_k-F_{k-1}} - (\frac{\rho}{k})^{F_k-R_{k-1}+1} \right], \\ \qquad\qquad \text{if } m = F_k \,. \end{cases}$$

Note that $\pi_k(R_{k-1}+1)$ is determined through normalisation condition which states that the sum of probabilities of all states of the micro-chain of level k equals 1. So we obtain :

$$\pi_k(R_{k-1}+1) = \left(\sum_{m=R_{k-1}+1}^{F_k} \gamma_m^k \right)^{-1} .$$

We notice, in the equations above, that the stationary probability formulas of $\pi_k(m)$ of levels k only depend on the thresholds R_{k-1}, R_k, F_{k-1} and F_k. This shows that the variation of a threshold R_k or F_k has an impact only on levels k and $k+1$ but not on the other levels. Moreover, since the micro-chain of level k starts from state $(R_{k-1}+1, k)$ and ends in state (F_k, k), then thresholds from level 1 to $k-2$ as well as thresholds from level $k+2$ to K are not involved on the stationary distribution of the level k.

To resume, if we modify an activation threshold F_k, then it will modify states (F_k, k) and $(F_k+1, k+1)$ and the two micro-chains of level k and $k+1$. If we modify, a deactivation threshold R_{k-1}, then it will modify states $(R_{k-1}+1, k)$ and $(R_{k-1}, k-1)$ and the two micro-chains of levels $k-1$ and k.

Finally, the variation of a threshold from level k modifies only the two micro-chains of level k and $k+1$. Hence, to compute the cost with respect to Theorem 1 we only need to recalculate the stationary probabilities and thus the associated costs of these levels. The costs of the other levels are left unchanged.

Impact on the macro-chain : A change in a threshold value modifies some transitions of the macro-chain. This can be seen, similarly as previously, using the balance equations :

The transition values are :

$$\begin{cases} \lambda_k &= \lambda \cdot \pi_k(F_k) & \forall\, k &= 1, \ldots, K-1 \\ \mu_k &= k \cdot \mu \cdot \pi_k(R_{k-1}+1) & \forall\, k &= 2, \ldots, K \end{cases} .$$

This gives the following formulas, for all $k \in \{2, \ldots, K\}$:

$$\Pi(k) = \Pi(1) \cdot \prod_{j=1}^{k-1} \left(\frac{\lambda_j}{\mu_{j+1}} \right) ,$$

and, from the normalisation condition, we get :

$$\Pi(1) = \left[1 + \sum_{k=2}^{K} \prod_{j=1}^{k-1} \left(\frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1} .$$

Since $\Pi(1)$ depends on all threshold values and since we notice that $\Pi(k)$ depends on $\Pi(1)$ for all $k \in \{2, \cdots, K\}$, therefore, when we modify a threshold F_k or R_k of any level k we need to recalculate the whole stationary probabilities of all states of the macro-chain. □

4.4.2.1 Cost calculation with aggregation technique

The aggregation framework described above, allows to compute the global cost $\overline{C}^{\pi}_{[F,R]}$, defined by Eq. (4.3), in an aggregated form based on the costs per level computed with micro-chains. Algorithm 8 displays how to compute the global cost with a given set of initial thresholds denoted by $[F, R]_0$. Algorithm 9 displays how to compute the global cost with a given set of thresholds $[F, R]$ after an iteration process where one of the thresholds has changed (either F_k or R_k).

Algorithm 8: First aggregated cost computation : FACC

Input: $[F, R]_0, \lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: $C^{[F,R]_0}$

/* Calculation of stationary distributions of micro-chains and costs per level */
for $1 \leq l \leq K$ **do**
  Generate matrix P^l of the level l micro-chain and compute stationary vector π_l
  $C(l) = 0$ // Cost per level
  for $(\, m = R_{l-1} + 1 \,;\, m \leq F_l \,;\, m++)$ **do**
    Compute $\pi_l(m) * C(m, l)$
    $C(l) = C(l) + \pi_l(m) * C(m, l)$
/* Calculation of the stationary distribution of the macro-chain and the global cost */
Build the generator matrix $\mathcal{Q}$ of the macro-chain from the local micro-chains π_l and compute the stationary vector Π
$C = 0$ // Global cost
for $1 \leq l \leq K$ **do**
  Compute $\Pi(l)$
  $C = C + C(l) * \Pi(l)$

Algorithm 9: Calculation of the cost for the current solution after changes in F_l or R_l : CACC

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: $C^{[F,R]}$

```
/* Calculation of the stationary distributions of the micro-chains and the costs per level
   impacted by the modified thresholds                                                    */
```
1 **for** $l = k\,,\ k+1$ **do**
2 Generate matrix P^l of the level l micro-chain and compute stationary vector π_l
3 $C(l) = 0$ // Cost per level
4 **for** $(m = R_{l-1}+1\,;\ m \leq F_l\,;\ m++)$ **do**
5 Compute $\pi_l(m) * C(m,l)$
6 $C(l) = C(l) + \pi_l(m) * C(m,l)$

```
/* Calculation of the stationary distribution of the new macro-chain and the global cost  */
```
7 Build the generator matrix $\mathcal{Q}$ of the macro-chain from the local micro-chains π_l and compute the stationary vector Π
8 $C = 0$ // Global cost
9 **for** $1 \leq l \leq K$ **do**
10 Compute $\Pi(l)$
11 $C = C + C(l) * \Pi(l)$

4.5 Heuristics for thresholds calculation using stationary distributions of Markov chain

Now that we have presented the SCA method for the steady-state computation, we present in this section the different optimisation methods in order to minimise the global cost function. Our first approach relies on thresholds calculation using the stationary distribution of the underlined Markov chain coupled with a deterministic optimisation problem. We focus here on different deterministic algorithms which compute the optimal threshold policy minimising the expected global cost.

The optimisation problem For a set of fixed parameters : λ, μ, B, K, and costs : C_a, C_d, C_h, C_s, C_r, the algorithms seeks to find the vector of thresholds $[F, R]$, such that the objective function is minimal. The objective function is the expected cost $\overline{C}^{\pi}_{[F,R]}$ defined by Eq.(4.3). For any solution $[F, R]$, the computation of the expected cost depends on the stationary distribution π of the Markov chain $\{X(t)\}_{t\geq 0}$ induced by $[F, R]$. The thresholds must verify some given constraints resulting from the principle of hysteresis. That is : $R_k < F_k$, $R_{k-1} < R_k$ and $F_{k-1} < F_k$. Also, it is assumed that servers do not work for free. That is : $k < F_k$. Therefore, we have to solve the constrained optimisation problem given by :

$$\begin{aligned} \underset{F,R}{\text{Minimise}} \quad & \overline{C}^{\pi}_{[F,R]} \\ \text{u.c.} \quad & R_i < F_i \quad i = 1, \ldots, K-1. \\ & F_1 < F_2 < \cdots < F_{K-1} < B \cdot \\ & 0 \leq R_1 < R_2 < \cdots < R_{K-1} \\ & F_i\,,\ R_i \in \mathbb{N} \end{aligned} \tag{4.7}$$

Obviously, this optimisation problem seems NP-hard, as the cost function is non-convex (see Section 4.5.4.1)

and the resolution time increases exponentially with the number of thresholds.

4.5.1 Local search heuristics

To solve this optimisation problem, several local search heuristics have been developed in [141]. Most of the heuristics are inspired by Kranenburg et al. [84] which presents three different heuristics to resolve a lost sales inventory problem with, unlike us, only a single threshold by level. Actually usual base-stock heuristics had to be adapted to the hysteresis model as it is necessary to manage double thresholds now. To detail the different algorithms, we define :

- $[F, R]$: vector of thresholds.
- *solve*$[F, R]$: function that, for a given vector of thresholds, generates its associated Markov Chain, computes its stationary distribution with the aggregated chain method then computes the mean cost $\overline{C}^{\pi}_{[F,R]}$ from Eq. (4.3).
- C^* is the solution of the optimisation problem (4.7) and $[F, R]^*$ is the corresponding vector of thresholds.

These heuristics are differentiated by several points : the initialisation phase, the choice of the neighbourhood and how we explore different threshold vectors. This has a noticeable influence on the convergence of the algorithms that we illustrate later in numerical experiments (see section 4.8). It could be noticed that the running time of algorithms will depend both on the number of threshold vectors they will explore during the execution, but also on the time to calculate the stationary probability of micro-chains and macro-chain. This time is not constant since it increases with the system load ρ. We briefly detail the three heuristics in an intuitive way to understand their behaviour, then display the algorithms.

Best Per Level (**BPL**) This algorithm first initialises $[F, R]$ with the lowest feasible value, i.e. $F_1 = 1, F_2 = 2, ..., F_{K-1} = K$ and $R_1 = 0, R_2 = 1, ..., R_{K-1} = K - 1$. Then it improves each threshold in the following order : it starts with the first activation threshold F_1 by testing all its possible values taking into account the hysteresis constraints. A new value of F_1 that improves the global cost, will replace the old one. Then, it will move on to $F_2, F_3, ..., F_{K-1}, R_1, R_2, ..., R_{K-1}$. Once a loop is finished, it will restart again until the mean global cost is not improved anymore (see details in Algorithm 10).

Algorithm 10: Best per Level

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$

Initialise a vector of thresholds $[F, R]_0$
Apply **solve**$[F, R]$ for the current vector
Repeat
`/* Activation Thresholds */`
for $k \in [1, K-1]$ **do**
  for F_k respecting constraints **do**
    solve$[F, R]$ and **store** the vector if improvement
`/* Deactivation Thresholds */`
for $k \in [1, K-1]$ **do**
  for R_k respecting constraints **do**
    solve$[F, R]$ and **store** the vector if improvement
Until no improvement in a loop

Increasing (**INC**) The process of this heuristic is to increase by one a threshold (F_k or R_k) when testing a new feasible solution. When this improves the solution we keep the new value otherwise we keep the old one. Then we move to a next set of thresholds by increasing one of them by one (see details in Algorithm 11).

Algorithm 11: Increasing

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$
Initialise a vector of thresholds $[F, R]_0$
Apply **solve**$[F, R]$ for the current vector
Repeat
`/* Activation Thresholds */`
for $k \in [1, K-1]$ **do**
  Give +1 to F_k, **solve**$[F, R]$ and **store** the vector if improvement
`/* Deactivation Thresholds */`
for $k \in [1, K-1]$ **do**
  Give +1 to R_k, **solve**$[F, R]$ and **store** the vector if improvement
Until no improvement in a loop

Neighborhood Local Search (**NLS**) This algorithm is the classical local search algorithm. We randomly initialise the solution $[F, R]$. Then we generate the neighborhood $\mathcal{V}(x)$ of a current solution x. Each neighboring solution is the same as the current solution except for a shift ± 1 for one of the thresholds. The algorithm explores all the neighborhood and returns the best solution among $\mathcal{V}$. Again, it loops the same process until the mean global cost is not improved anymore (see details in Algorithm 12).

Algorithm 12: Neighbour Local Search

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$
Initialise a vector of thresholds $[F, R]_0$
Apply **solve**$[F, R]$ for the current vector
Repeat
Generate the neighbourhood **V**
for $[F, R] \in \mathbf{V}$ **do**
  solve$[F, R]$ and **store** the vector if improvement
Until no improvement in a loop

4.5.2 Improvement of local search algorithms with aggregation

Local search algorithms are based on an exploration step in a set of feasible solutions that must be evaluated. From section 4.3.3, the evaluation of a solution depends on the stationary distributions of micro-chains and the macro-chain as well as the costs per level, thus we should, after any threshold modification, compute them again. But the use of Corollary 2 will widely improve the algorithm's speed without impacting the efficiency. Indeed, it allows to compute only the stationary distributions of the two impacted micro-chains k, $k+1$ among the K micro-chains and their associated costs. The macro chain still needs to be solved.

Although decomposition and aggregation approaches reduce the number of computations to perform, some algorithmic tricks should be used in order to ensure a proper running of the method. we have to store the two modified micro-chains of the current solutions since they will be used many times. The modification of local search heuristics with the help of aggregation reduces the number of computations to perform and then the running time. Moreover, we can conceive that this method will be more effective as K increases.

We provide an illustration with **BPL algorithm** coupled with aggregation in algorithm 13. The functioning is the same for all local search heuristics and we will refer to coupled heuristics with aggregations as **BPL Agg**, **INC Agg** and **NLS Agg**. The main change comes from the fact that in the aggregated version algorithms, the cost is computed with the FACC method at the beginning and then with CACC during the rest of the algorithm which drastically reduces the complexity of the methods.

Algorithm 13: Best per Level coupled with cost aggregation

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$
Initialise a vector of thresholds $[F, R]_0$
Apply **solve**$[F, R]$ with **FACC** for the current vector
Repeat
`/* Activation Thresholds */`
for $k \in [1, K-1]$ **do**
    for F_k respecting constraints **do**
        solve$[F, R]$ with **CACC** and **store** the vector if improvement
`/* Deactivation Thresholds */`
for $k \in [1, K-1]$ **do**
    for R_k respecting constraints **do**
        solve$[F, R]$ with **CACC** and **store** the vector if improvement
Until no improvement in a loop

4.5.3 Improvement of local search heuristics coupled with initialisation techniques

As it stands, the proposed local search heuristics do not have specific initialisation process and one possibility to improve them is to quickly compute a reasonable initial solution to accelerate convergence. We display here two heuristics for the initialisation : *Fluid approximation* and *MMK approximation*.

4.5.3.1 Initialisation with fluid approximation

Mitrani in [103] proposed a multi server system with a fixed amount of servers that are always turned on and a reserve of servers that can be activated if the load of the system increases, or deactivated if the load decreases. In this work the author proposed a heuristic based on fluid approximations to find the optimal threshold for activation and the one for deactivation. This corresponds to a two-levels model with only one activation threshold and one deactivation threshold. This is done in order to minimise a cost which includes an holding cost and a server utilisation cost. However, since we consider a more complex cost function (activation, deactivation and reject cost) and a more complex model (multilevel thresholds), we have to adapt this fluid approximation method to our approach. Since we turn on VMs one by one, we need to find an optimal threshold at each level which indicates when to turn on or down a VM depending on the number of requests in the system.

We adapt this heuristic in our approach as follows :

Deactivation Thresholds Similarly as [103], we decide to turn off virtual machines whenever they are unused as we don't want VMs to work for free. For the deactivation thresholds, we consider at each level k that the reserve is composed of only one virtual machine, since we have k VMs activated. This implies that we have $k-1$ VMs activated plus the reserve, which is exactly one VM here. Therefore, deactivating the reserve will lead to $R_k = k-1$.

Activation Thresholds Adding a new VM will empty the queue faster and indeed will improve the QoS, but will deteriorate the energy part by adding operational costs. Letting unchanged the current number of activated VMs will empty the queue in a slower time but will be less expensive regarding energy costs. In order to obtain activation thresholds, we use a fluid approximation (inspired from [103]) to approximate if it is useful to turn on a new VM to empty the queue or not. We define by $\tilde{C}_k(u)$ the approximated cost corresponding to k working servers and u requests is the system :

$$\tilde{C}_k(u) = \frac{u}{2} \cdot \tilde{B}_k(u) \cdot C_h + k \cdot \tilde{B}_k(u) \cdot C_s + \tilde{p}_k \cdot \frac{\lambda}{\lambda + k \cdot \mu} \cdot C_r \,. \tag{4.8}$$

The time until the queue is empty when there are u requests in the system, is defined by $\tilde{B}_k(u)$. Assuming the queue behaves like a deterministic fluid, the load would decrease at the rate of $k \cdot \mu - \lambda$ and we get

$$\tilde{B}_k(u) = \frac{u}{k\mu - \lambda} \text{ unit of times} \,.$$

The loss probability in level k is $\tilde{p}_k$ and is computed by approximating the loss probability of an M/M/1/B queue, with arrival rate λ, and service rate $k \cdot \mu$. Notice that $(u \cdot C_h \cdot \tilde{B}_k(u))/2$ represents the queuing cost and $\tilde{B}_k(u) \cdot k \cdot C_s$ is the consumption cost of k active VMs.

The activation threshold F_k represents the smallest amount of requests u in the system, for which it is interesting to activate a new VM, i.e. the cost for $k+1$ VMs activated plus the activation cost is lower than the cost with k VMs activated. At level k, we thus solve a computational problem to determine the activation thresholds. Precisely, we have to find the first u $\in \mathbb{N}$ denoted by $\overline{u_k}$ such that :

$$\tilde{I}(u) = \tilde{C}_{k+1}(u) + C_a - \tilde{C}_k(u) < 0 \,.$$

However, $\tilde{B}_k(u)$ is relevant only when $k\mu - \lambda > 0$, which intuitively means that the queue can be empty. Otherwise if $\lambda \geq k\mu$, we decide, in those cases, to fix the thresholds as low as possible, which is relevant because if the load is high, then one has to activate the VMs quickly to satisfy the QoS defined by the SLA. Finally, the optimal activation thresholds are given by :

$$F_k = \begin{cases} \max\{F_{k-1}+1, \min\{\overline{u_k}, B\}\} & \text{if } \lambda < k\mu \\ R_k + 1 & \text{if } \lambda \geq k\mu \end{cases} \,.$$

The pseudo-code is given in the Algorithm 14.

Algorithm 14: Fluid approximation

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$
/* Deactivation Thresholds */
for $k \in [1, K-1]$ **do**
  $R_k = k - 1$
/* Activation Thresholds */
for $k \in [1, K-1]$ **do**
  while $\tilde{I}(u) > 0$ **do**
    for $u \in [F_{k-1} + 1, B]$ **do**
      Compute $\tilde{I}(u)$
  $\bar{u}_k$ = last u computed
  $F_k \leftarrow \bar{u}_k$ w.r.t. hysteresis constraints

4.5.3.2 Fluid approximation coupled with local search heuristics

It will be observed later (section 4.8) that **NLS** is not efficient mainly because of its initialisation phase in which thresholds are randomly chosen. Since fluid approximation heuristic is very fast but also not so efficient as we will see in the experimental results of section 4.8, our aim is to improve both speed and accuracy of the local search heuristics by coupling them. The initialisation phase of the **NLS** will be now performed by the fluid approximation method. This initialisation is very fast and we try to be placed in the basin of attraction of the optimal solution that we will be reached with the local search. We define by **NLS Fluid** the adapted heuristic (see pseudo-code 15).

Algorithm 15: Neighbour Local Search with Fluid approximation initialisation

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$
Initialise a vector of thresholds $[F, R]_0$ with **Fluid approximation**
Apply **solve**$[F, R]$ for the current vector
Repeat
Generate the neighbourhood **V**
for $[F, R] \in$ **V do**
  solve$[F, R]$ and **store** the vector if improvement
Until no improvement in a loop

4.5.3.3 Initialisation with queuing model approximations

We work here on a new heuristic which calculates a near-optimal solution very quickly. It can be considered as a method in its own right but can be also used for the initialisation step in local search heuristics. This heuristic is called **MMK approximation** since it uses $M/M/k/B$ results to compute costs of these fairly close models. We devised this heuristic because we obtained promising results with the Fluid approximation for initialising the **NLS** algorithm. Yet we think that we can do better initialising the heuristics with an other method to approximate the first solution.

Principle of the heuristic The main idea of the heuristic is that, at each level k, we compute an approximation of the mean global cost by using the stationary distribution of the Markov chain of a $M/M/k/B$ queue model instead of using the Markov chain of Section 4.3.2. In such queues, the stationary distribution of the Markov chain is given by a closed formula. Thus, the computation of the distribution is done in a constant time due to the closed formula while the exact computation relative to the stationary distribution of the Markov chain is much longer. Hence, we obtain an approximate solution of the expected cost very quickly. In order to find the best threshold m for which it is better to activate or deactivate, we proceed by comparing approximated costs of having k VMs activated with the one of having $k+1$ (respectively $k-1$) VMs activated added by the activation (respectively deactivation) cost. This indicates if it is worth being in level k or $k+1$ for a given number of requests m in the system. The detailed heuristic with calculations is described below.

Design of the heuristic We recall the formulas of the stationary distributions in $M/M/k/B$ queues. Let $\hat{\pi}(k,m)$ be the stationary distribution of state (m,k), we get :

$$\hat{\pi}(k,m) = \hat{\pi}(k,0) \cdot \frac{\rho^m}{m!} \text{ for } 1 \leq m \leq k \quad \text{and} \quad \hat{\pi}(k,m) = \hat{\pi}(k,0) \cdot \frac{a^m \cdot k^k}{k!} \text{ for } k < m \leq B.$$

where $\rho = \dfrac{\lambda}{\mu}$, $a = \dfrac{\rho}{k} < 1$ and where $\hat{\pi}(k,0)$ is :

$$\hat{\pi}(k,0) = \left(\sum_{m=0}^{k-1} \frac{\rho^m}{m!} + \frac{k^k}{k!}(B-k+1) \right)^{-1} \text{ if } a = 1 \text{ and } \hat{\pi}(k,0) = \left(\sum_{m=0}^{k-1} \frac{\rho^m}{m!} + \frac{\rho^k}{k!} \frac{1-a^{B-k+1}}{1-a} \right)^{-1} \text{ if } a \neq 1.$$

We define $\hat{C}_k(m)$ as the approximate cost of having k VMs turned on and m requests in the system by :

$$\hat{C}_k(m) = \hat{\pi}(k,m)\big(m \cdot C_H + k \cdot C_S\big) + \hat{\pi}(k,B) \cdot \lambda \cdot C_R \,;$$

$\hat{C}^A_{k+1}(m)$ as the approximate cost of having m requests in level $k+1$ knowing that we have just activated a new VM by :

$$\hat{C}^A_{k+1}(m) = \hat{\pi}(k+1,m)\big(m \cdot C_H + (k+1) \cdot C_S\big) + \hat{\pi}(k+1,B) \cdot \lambda \cdot C_R + \hat{\pi}(k,m-1)\lambda C_A \,.$$

Last we define $\hat{C}^D_k(m)$ as the approximate cost of having m requests in level k knowing that we have just deactivated a new VM by :

$$\hat{C}^D_k(m) = \hat{\pi}(k,m)\,(mC_H + kC_S) + \hat{\pi}(k,B)\lambda C_R + \hat{\pi}(k+1,m+1)\min(m+1,k+1)\mu C_D \,.$$

We want to compare these costs to know whether we should activate, deactivate or let the servers unchanged.

Activation thresholds . We define $\phi^A_k(m) = \hat{C}_k(m) - \hat{C}^A_{k+1}(m)$. If $\phi^A_k(m) < 0$ then it is better to not activate a new server while if $\phi^A_k(m) \geq 0$ it is better to activate a new one. There is no evidence of the monotonicity of $\phi^A_k(m)$ in m, nevertheless we choose the threshold by $F_k = \min\big\{m : \phi^A_k(m) \geq 0\big\}$. To compute the whole thresholds, all k are studied in an ascending order. For a fixed level k, we need to compute $\phi^A_k(m)$ for all m and stop when the function $\phi^A_k(m)$ is larger than 0.

Deactivation thresholds . We define $\phi_k^D(m) = \hat{C}_{k+1}(m) - \hat{C}_k^D(m)$. If $\phi_k^D(m) < 0$ then it is better to stay with $k+1$ servers while if $\phi_k^D(m) \geq 0$ it is better to deactivate a virtual machine. Similarly, there is no evidence about the monotonicity of $\phi_k^D(m)$ but we define $R_k = \min\left\{m : \phi_k^D(m) \geq 0\right\}$. The computation of the whole deactivation thresholds is similar to the activation ones.

We summarise the **MMK approximation** heurisitc in pseudo-code 16.

Algorithm 16: MMK approximation

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$

/* Activations thresholds */
Activation Threshold 1st Level
k=1 // First level
for $m \in [1, B]$ **do**
  Compute $\phi_1^A(m)$
  When $\phi_1^A(m) > 0$,BREAK, then F_1 = m if constraints respected
Activation thresholds level k **for** $k \in [2, K-1]$ **do**
  for $m \in [F_{k-1} + 1, B]$ **do**
    Compute $\phi_k^A(m)$
    When $\phi_k^A(m) > 0$,BREAK, then F_k = m if constraints respected
/* Desactivations thresholds */
Deactivation Threshold 1st Level
k=1 // First level
for $m \in [0, F_k - 1]$ **do**
  Compute $\phi_1^D(m)$
  When $\phi_1^D(m) \geq 0$,BREAK, then R_1 = m if constraints respected
Deactivation Thresholds level k **for** $k \in [2, K]$ **do**
  for $m \in [R_{k-1} + 1, F_k - 1]$ **do**
    Compute $\phi_k^D(m)$
    When $\phi_k^D(m) \geq 0$,BREAK, then R_k = m if constraints respected

Specific case where not all levels are reached When we use this method to compute the optimal vector of thresholds, it appears that the optimal policy returned by the heuristic does not activate or deactivate every VMs depending on the system load. Under specific loads, it can appear that none m values satisfy inequality on ϕ. Thus the heuristic does not find activation or deactivation thresholds for some levels k. In scenarios where the load is very low and where the heuristic does not find some activation thresholds for all levels : we define F_k = B - k + 1 and R_k = k. Similarly considering the case where the load is too high and where some deactivation thresholds are not all computed for all levels : we defineF_k = k and R_k = B-K.

This is discussed in more details in section 4.6.

4.5.3.4 MMK approximation coupled with local search heuristics

The initialisation of the local search heuristics presented in 4.5.1 either is totally random or simply chooses the bounds (lowest values or highest values) of the solution space. The aim here is to improve the speed and the accuracy of these heuristics by coupling them with the **MMK approximation** used as an initialisation step. This initialisation aims at finding an initial solution that will be in the basin of attraction of the optimal solution.

Starting with this solution improves the ability of local search algorithms to reach the best solution in a faster time. We define by **Alg MMK** the coupled heuristic with MMKB approximation (see pseudo-code 17).

Algorithm 17: Neighbour Local Search

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$
Initialise a vector of thresholds $[F, R]_0$ with **MMK approximation**
Apply **solve**$[F, R]$ for the current vector
Repeat
Generate the neighbourhood **V**
for $[F, R] \in$ **V do**
    solve$[F, R]$ and **store** the vector if improvement
Until no improvement in a loop

4.5.4 Meta-heuristic approach : the Simulated Annealing (SA)

Numerical experiments in Section 4.8 will show that local search heuristics can obtain local minima. To address this issue, we investigate meta-heuristics whose design is made to avoid local minima. This investigation is made to improve the accuracy of the solutions returned by local searches, while maintaining a reasonable execution speed. In this section, we first give an example showing the non-convexity of the cost function that explains why local search heuristics can fall into local optima. Next we provide the methodology of the *Simulated Annealing* (SA) which is one possible meta-heuristic and last we address its parameterisation.

4.5.4.1 Non-convexity of the cost function

First we point out the non-convexity of the cost function in the following example.

Example 2. *We took a low scale system ($K = 3$ and $B = 20$). We take $\lambda = 1$, $\mu = 1$, $C_a = 0.5$, $C_d = 0.5$, $C_s = 20$, $C_h = 20$, $C_r = 1$. The solution of the exhaustive search is the following : Optimal Cost = $52, 53$ with thresholds $[F_1, R_1, F_2, R_2] = [2, 1, 3, 2]$ and the one of the NLS is : Optimal Cost = $53, 17$ with thresholds $[F_1, R_1, F_2, R_2] = [1, 0, 3, 2]$. The NLS returned solution is not optimal and we then study its neighbourhood to see if it was indeed a local minimum. We studied all the possible neighbours (under the constraints) of the solution returned by the NLS and verified that their costs were indeed higher than the returned one.*

This counter-example shows that the cost function can be non-convex and therefore heuristics can be stuck in a local minimum. The use of the simulated annealing is thus relevant in our scenario.

4.5.4.2 Intuition of the Simulated Annealing

The simulated annealing (**SA**) [9] is a meta-heuristic that aims to avoid the convergence in a local minimum, allowing for certain iterations to exit this local minimum by accepting a higher cost or a deterioration of the objective function with a probability, defined by the Boltzmann probability distribution $\exp(-\frac{\triangle C}{T_i})$. T_i represents the current temperature and $\triangle C$ is the mean difference between the cost of the current solution and the costs of its neighbours. This is done at the expense of the execution time, which can be very long for the Simulated An-

nealing, to converge towards the global optimum. We give in the algorithm 18 the main steps of the SA approach.

Algorithm 18: Simulated Annealing

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$
Output: C^* , $[F, R]^*$
Initialise at random a vector of thresholds $[F, R]_0$, and optimal parameters $< T_0, \alpha, L >$
Repeat
Apply **solve**[F,R] for the current vector
Generate its neighbourhood $\mathbb{V}$
for $[F, R] \in \mathbb{V}$ **do**
    solve$[F, R]$, and compute the cost
    if Improvement **then**
        Save the solution
    else
        Draw a probability $p \in [0, 1]$
        if $p \leq e^{-\triangle C}/T_i$ **then**
            Accept the new solution
    Every L iterations, $T_{i+1} = \alpha T_i$, and go to (Step 2)
Until $T_i = 0$ OR Max Iterations reached

4.5.4.3 Simulated Annealing parameterisation

The principal challenge in meta-heuristics is to fix the parameters values of the algorithm as best as possible. Indeed, depending on the values of the initial parameters, the meta-heuristic will be accurate or not. Some guidelines for determining parameters have been given in [9] and we adapt them for our purpose. This requires some initial simulations to capture how the simulated annealing behaves in our case so one can tune its parameters optimally. The setting of the parameters is a trial and error process. Basically, one try arbitrary settings and launch the algorithm then adjust step by step the parameters. The initial temperature T_0 should be taken such that a lot of worse solutions will be saved at the beginning. Then it will follow a decreasing sequence such that at the end of the algorithm, it will only take neighbours improving the solution. The intuition is that when T is high enough compared to $\triangle C$, then we accept enough deteriorated solutions and as a reverse, when T is near 0, we will not accept deteriorated solutions anymore. One method to compute the initial temperature T_0 is to launch some simulations with random settings and take a mean cost difference $\overline{\triangle C}$ between a solution and its neighbours. Then we obtain :

$$T_0 = -\overline{\triangle C}/\ln(p_0)\,,$$

where p_0 is the initial probability generated such that the initial temperature T_0 will be high. The probability of taking a wrong solution at the beginning is high (≈ 0.9), and then we want to make this temperature decrease such that the accepting probability tends to 0 at the end and, by this way, we only choose better solutions.

Once the initial temperature is chosen, one may find the cooling factor $\alpha < 1$ which is the rate of the decreasing sequence $T_{i+1} = \alpha T_i$. Usually in the Simulated Annealing, it is fixed at 0.9 or 0.95 but in our case, it will be fixed very low, at 0.1, since we have small values for the cost. Indeed, since our value of T is very low, because $\overline{\triangle C}$ is small, if the decreasing rate is too large, then the temperature will decrease slowly and will reach 0 in a huge number of iterations. If α is very small, then the temperature decreases faster and we reach $T = 0$ in a shorter time, which leads to the end of the algorithm. This is a reason why taking a small value for α (like 0.1) is relevant in our setting. The last parameter to optimise is the number of iterations L per step of the temperature sequence :

it must be sufficiently large to accept enough deteriorated solutions.

4.5.5 Heuristics comparisons : concluding remarks

Finally, we have many heuristics that we compare in numerical experiments (see section 4.8) by their accuracy and execution time. To summarise (see table 4.1), we have the three main local search heuristics : BPL, INC and NLS. Next we have the aggregation cost technique that we couple with each heuristics. We also provide initialisation with Fluid approximation and MMK approximation for the heuristics. Last, we devise a meta-heuristics (simulated annealing).

TABLE 4.1 – Summary of algorithms based on Markov chain stationary distribution computation

Algorithms	Settings
BPL	Select the best threshold at each iteration
INC	Increase by one threshold at each iteration
NLS	Select the best neighbour at each iteration
BPL Agg	BPL with aggregation cost computation
INC Agg	INC with aggregation cost computation
NLS Agg	NLS with aggregation cost computation
Fluid	Fluid approximations for fast computation of set of thresholds
NLS-Fluid Agg	NLS with Fluid approximation initialisation
MMK	MMK queue approximation for fast computation of set of thresholds
BPL MMK	BPL with MMK approximation initialisation
INC MMK	INC with MMK approximation initialisation
NLS MMK	NLS with MMK approximation initialisation
Simulated Annealing	Meta-heuristic to avoid local optima

4.6 Computing policies with Markov Decision Process

We consider here the multi-server queue of Section 4.2.2 and the controlled model in which only a single virtual resource can be activated or deactivated at a time decision. This optimal control problem is a continuous time process and thus should be solved using a Semi Markov Decision Process (SMDP). In this section, we first describe the uniformised SMDP and its solving. Then we underline the differences between the optimal control model and the optimisation problem of Section 4.3.2.

4.6.1 The SMDP model

4.6.1.1 Elements of the SMDP

The state space is the one defined in Section 4.2 : $\mathcal{S} = \{0, 1, \ldots, B\} \times \{1, \ldots, K\}$. Hence, a state $s \in \mathcal{S}$ is such that $x = (m, k)$ where m is the number of customers in the system and k the number of active servers. Similarly, the action space $\mathcal{A} = \{-1, 0, 1\}$ represents respectively : deactivate one machine, left unchanged the active servers, or activate one machine.

The system evolves in continuous time and at some epoch a transition occurs. When a transition occurs, the controller observes the current state and reacts to adapt the resources by activating or deactivating the virtual

machines. The activation or deactivation are instantaneous and then the system evolves until another transition occurs. Two events can occur : an arrival with rate λ which increases the number of customers present in the system by one or a departure which decreases the number of customers by one.

Let $x = (m, k)$ be the state and a the action, we define $N(k + a)$ as the real number of active VM after the action a was triggered. We have $N(k+a) = \min\{\max\{1, k+a\}, K\}$. It follows that the transition rate is equal to

$$\begin{cases} \lambda & \text{if } y = \big(\min\{m+1, B\}, N(k+a)\big) \\ \mu \cdot \min\{m, N(k+a)\} & \text{if } y = \big(\max\{0, m-1\}, N(k+a)\big) \end{cases}.$$

4.6.1.2 Uniformised transition probabilities

We apply here the standard method to deal with continuous time MDP : the uniformisation framework. We follow the line of chapter 11 of [5] to define the uniformised MDP.

We define $\Lambda(m, k, a)$ as the transition rate per state. We have $\Lambda(m, k, a) = \lambda + \mu \cdot \min\{m, N(k+a)\}$. From now on, any component denoted by "$\sim$" refers to the uniformised process. We thus define the uniformisation rate by $\tilde{\Lambda}$ with $\tilde{\Lambda} = \max_{(m,k,a)} \Lambda(m, k, a) = \lambda + K\mu$ which is the maximum transition rate. The transition probability from state x to state y when action a is triggered is denoted by $\tilde{p}(y|x, a)$ in the uniformised model.

We have :

$$\tilde{p}(y|x,a) = \begin{cases} \dfrac{\lambda}{\tilde{\Lambda}} & \text{if } y = (m+1, N(k+a)) \\ \dfrac{\mu \min\{m, N(k+a)\}}{\tilde{\Lambda}} & \text{if } y = (m-1, N(k+a)) \\ \dfrac{\tilde{\Lambda} - \Lambda(m,k,a)}{\tilde{\Lambda}} & \text{when } y = (m,k) \\ 0 & \text{otherwise,} \end{cases}$$

when $x = (m, k)$ such that $0 < m < B$ and a is arbitrary; when $x = (B, k)$ with $a \neq 0$, or when $x = (B, k)$ with $k \neq K$ and $a \neq +1$ or also when $x = (B, k)$ with $k \neq 1$ and $a \neq -1$; when $x = (1, k)$ with $a \neq 0$, or when $x = (1, k)$ with $k \neq K$ and $a \neq +1$, or also when $x = (1, k)$ with $k \neq 1$ and $a \neq -1$.

We have :

$$\tilde{\Lambda} \times \tilde{p}(y|x,a) = \begin{cases} \mu \min\{m, N(k+a)\} & \text{if } y = (B-1, N(k+a)) \\ \tilde{\Lambda} - \mu \min\{m, N(k+a)\} & \text{when } y = (B,k) \\ 0 & \text{otherwise,} \end{cases}$$

when $x = (B, k)$ with $a = 0$, or when $x = (B, k)$ with $k \neq K$ and $a \neq +1$, or also when $x = (B, k)$ with $k \neq 1$ and $a \neq -1$.

We have :

$$\tilde{\Lambda} \times \tilde{p}(y|x,a) = \begin{cases} \lambda & \text{if } y = (1, N(k+a)) \\ \tilde{\Lambda} - \lambda & \text{when } y = (0,k) \\ 0 & \text{otherwise,} \end{cases}$$

when $x = (0, k)$ with $a = 0$, or when $x = (0, k)$ with $k \neq K$ and $a \neq +1$, or when $x = (0, k)$ with $k \neq 1$ and $a \neq -1$.

4.6.1.3 The uniformised stage costs

We take the definition of costs of Section 4.2.2. Instantaneous costs are charged only once and are related to activation, deactivation and losses. Accumulated costs are accumulated over time and are related to consumption and holding cost.

$$k(x,a) = \left(C_A \mathbb{1}_{\{a=1\}} + C_D \mathbb{1}_{\{a=-1\}}\right) + C_R.\mathbb{1}_{\{m=B\}} \text{ and } r(x,a) = \left(N(k+a)\cdot C_S + m \cdot C_H\right).$$

$$\forall x \in \mathcal{S}\ ,\ c(x,a) \ = \ k(x,a) \ + \ E\int_0^{\tau} r(x,a).dt$$

where τ is a *r.v.* following an exponential law of parameter $\Lambda(x,a)$

$$\tau \sim \mathbb{E}(\Lambda(x,a))$$

So, we have :

$$\forall x \in \mathcal{S}\ ,\ c(x,a) \ = \ k(x,a) \ + \int_0^{\infty}\int_0^{z} r(x,a).\Lambda(x,a).\exp^{-\Lambda(x,a)z}.dt.dz$$

$$\forall x \in \mathcal{S}\ ,\ c(x,a) \ = \ k(x,a) \ + \ r(x,a).\int_0^{\infty}\int_0^{z} \Lambda(x,a).\exp^{-\Lambda(x,a)z}.dt.dz$$

$$\forall x \in \mathcal{S}\ ,\ c(x,a) \ = \ k(x,a) \ + \ r(x,a).\int_0^{\infty}\int_0^{z} dt.\Lambda(x,a).\exp^{-\Lambda(x,a)z}.dz$$

$$\forall x \in \mathcal{S}\ ,\ c(x,a) \ = \ k(x,a) \ + \ r(x,a).\underbrace{\int_0^{\infty} z.\Lambda(x,a).\exp^{-\Lambda(x,a)z}.dz}_{=1\ /\ \Lambda(x,a) \text{ because } \mathrm{E}[\tau]}$$

Finally, we obtain [5] :

$$\boxed{\forall x \in \mathcal{S}\ ,\ c(x,a) \ = \ k(x,a) \ + \ r(x,a).\frac{1}{\Lambda(x,a)}} \tag{4.9}$$

For the treatment of stage costs, the non uniformised instantaneous costs and accumulated costs should be multiplied by $(\Lambda(x,a)/\tilde{\Lambda})$ to get the uniformised costs.

$$\tilde{k}(x,a) = \left(C_A \mathbb{1}_{\{a=1\}} + C_D \mathbb{1}_{\{a=-1\}}\right)\frac{\Lambda(x,a)}{\tilde{\Lambda}} + \frac{\lambda}{\tilde{\Lambda}} C_R.\mathbb{1}_{\{m=B\}} \text{ and } \tilde{r}(x,a) = \frac{1}{\tilde{\Lambda}}\left(N(k+a)\cdot C_S + m \cdot C_H\right).$$

$$\tilde{c}(x,a) \ = \ \tilde{k}(x,a) \ + \ \tilde{r}(x,a)\,.$$

After uniformisation of instantaneous and accumulated costs (Chapter 11.5 of [5]), we obtain the following equation of the uniformised stage cost for $x = (m, k)$:

$$\tilde{c}(x,a) = \big(C_A \mathbb{1}_{\{a=1\}} + C_D \mathbb{1}_{\{a=-1\}}\big)\frac{\Lambda(x,a)}{\tilde{\Lambda}} + \frac{\lambda}{\tilde{\Lambda}} C_R \mathbb{1}_{\{m=B\}} + \frac{1}{\tilde{\Lambda}}\big(N(k+a)\cdot C_S + m \cdot C_H\big)\,.$$

Objective function The objective function defined in Equation (4.1) translates for all $x \in \mathcal{S}$ into

$$\rho^{\pi}(x) = \lim_{N\to\infty} E^{\pi}\Big[\frac{1}{N}\sum_{t=0}^{N-1} \tilde{c}(x_t, \pi(x_t)) \mid x_0 = x\Big]\,,$$

for a given policy π. The value ρ^{π} is the expected stage cost and equals $\bar{C}$ when actions follow policy π.

4.6.2 Solving the MDP

4.6.2.1 Classification of the SMDP

Our SMDP is an average cost model. This is why, the expected stage costs depend on the recurrent properties of the underlying Markov chain that is generated by a deterministic policy. A classification is then necessary to study them. We use the classification scheme of [5]. For communicating multichain SMDP, [67] insures that uniformisation does not break neither structural properties of the generator nor the optimality of the policy computed in the uniformised model.

Definition 7 (Chapter 8.3.1 in [5]). A MDP is :

i Unichain if, the transition matrix corresponding to every deterministic stationary policy is unichain, that is, it consists of a single recurrent class, plus a possibly empty set of transient states;

ii Multichain if, the transition matrix corresponding to at least one deterministic stationary policy contains two or more recurrent classes;

iii Communicating if, for every pair of state x and y in $\mathcal{S}$, there exists a deterministic stationary policy π under which y is accessible from x, that is, $p_{\pi}^{n}(y,x) > 0$ for some $n \geq 1$;

Proposition 2. *The MDP is multichain. There is a stationary deterministic policy with monotone hysteresis properties that induces a corresponding Markov chain with more than two different recurrent classes.*

Proof. We assume that $K \geq 2$ and let k be such that $k \in [1, \dots, K]$. We define the policy q as follows. For any level l such that $l < k-1$, we have only activation. This means that there exists $m \in [0, \cdots, B]$ such that $q(m', l) = 1$ for any $m \leq m'$ and $q(m,l) \leq q(m, l-1)$. For level k we have neither activation nor deactivation and $q(m,k) = 0$ for all $m \in [0, \cdots, B]$. For any level l with $l > k+1$, we have only deactivation. There exists $m \in [0, \cdots, B]$ such that $q(m', l) = -1$ for any $m' \leq m$ and $q(m,l) \geq q(m, l+1)$.

Hence, starting from any state such that $l < k-1$, then the highest attainable level is $k-1$. Equally, starting from any state of the level k, we remain in this level. At last, starting from any state such that $l \geq k+1$, then the lowest attainable level is $k+1$. Therefore, we have three recurrent classes : the level k, the level $k-1$ and the level $k+1$ (and two recurrent classes for $K = 2$). □

We display an example in Figure 4.8 of an isotone hysteresis multichain MDP.

Lemma 3. *The MDP is communicating. There exists a stationary isotone hysteresis policy such that the corresponding Markov chain is irreducible.*

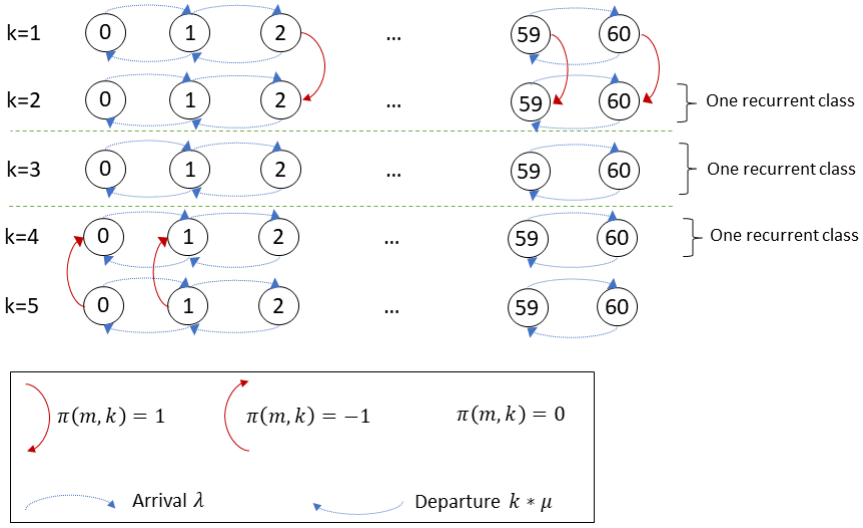


FIGURE 4.8 – Example of an isotone hysteresis multichain MDP

Proof. We exhibit such a policy. Let π be the deterministic stationary policy such that the thresholds l_k are defined by $l_k = 0$ and the thresholds L_k by $L_k = B$ for all k. The induced Markov chain is irreducible since any level can be reached from another one (when $m = 0$ or $m = B$) and since in a given level all the states are reachable from any state. Thus, we have that $p_\pi^n(y,x) > 0$ for some $n \geq 1$ for all couples (x,y). Therefore the MDP is communicating. □

Bellman Equations In the multichain case, the Bellman Equations are composed by two equations. In the uniformised model, we then have the two following optimality equations [5] :

$$\min_{a\in\mathcal{A}}\left\{\sum_{y\in\mathbb{X}}\tilde{p}(y\,|\,x,a)\rho(y)-\rho(x)\right\}=0 \text{ and } U(x)=\min_{a\in B_x}\left\{\tilde{c}(x,a)-\frac{\rho(x)}{\tilde{\Lambda}}+\sum_{y\in\mathbb{X}}\tilde{p}(y\,|\,x,a)\cdot U(y)\right\}$$

for all $x \in \mathcal{S}$, where

$$B_x=\left\{a\in\mathcal{A}\,\middle|\,\sum_{y\in\mathbb{X}}\tilde{p}(y\,|\,x,a)\rho(y)\;=\;\rho(x)\right\}.$$

It could be noticed that in the unichain case, $B_x = \mathcal{A}$ and that the two equations reduce to only the second one. These two non linear equation systems should be numerically solved to find $U(x)$ and $\rho(x)$. Once these terms are approximated we deduce the optimal policy with :

$$q(x)=\arg\min_{a\in B_x}\left\{\tilde{c}(x,a)+\sum_{y\in S}\tilde{p}(y\,|\,x,a)\cdot U(y)\right\}.$$

4.6.2.2 Algorithms

We describe here the choice of the algorithms used to solve this multichain and communicating model. Computations in a multichain model are far more complicated since testing if for any induced Markov chain there are several separated connected components seems to be a NP complete problem [5]. We first show that we actually can use some unichain algorithms due to the communicating properties of our SMDP, then we present two structured algorithms based on hysteresis properties.

Unichain algorithms From [5] it exists a multichain policy iteration algorithm that solves multichain models. It requires to solve two equation systems and thus is time consuming. However, by Proposition 3, our MDP is communicating and in Theorem 8.3.2 of [5] it is proved that unichain value iteration algorithm converges to the optimal value in communicating models. This property allows us to use the unichain value iteration algorithm.

There is also, in [5], a policy iteration algorithm for communicating models. It requires to start with an initial unichain policy and its inner loop roughly differs from the unichain case in order to keep some properties. We decide to use here algorithms based on unichain policy iteration. There does not exist theoretical guarantee of their convergence, but we showed in numerical experiments that they always converge to the same policy than value iteration.

Four different usual unichain algorithms will be considered : *Value Iteration*, *Relative value iteration*, *Policy Iteration modified* and *Policy Iteration modified adapted*, which adapts its precision during the policy evaluation step. They are respectively referred by *VI*, *RVI*, *PI* and *PI Adapt*. The algorithms are all described in [5] and are already implemented in the software [66].

Structured Policies algorithm We now integrate hysteresis properties in the algorithms. Two classes of policies have been investigated : Double Level class (Definition 2) and Monotone Hysteresis class (Definition 5). The goal is to plug hysteresis assumptions during the policy improvement step of policy iteration. This allows to test less actions at each iteration and to speed up the algorithm. Two algorithms are implemented : one for the double level properties (referred as *DL-PI*) and one for the hysteresis properties (referred as *Hy-PI*). We describe in pseudocodes 19 and 20 the modified policy iteration algorithms with integration of structural properties.

The properties of Double Level and Hysteresis policies induce some constraints on the best action selection in the Policy improvement step, mainly by restricting the choice of possible actions :

- Double Level : From Definition 2, the rule (policy) $q(m, k)$ is increasing in the number of jobs in the system m. This increasing property is integrated such that when we choose the best action, the set of decision is constrained. As an example : suppose that $q(0, k) = 0$, then $q(1, k) \geq 0$ which impose to test only two actions over three.
- Hysteresis : The hysteresis property integrates the Double Level property plus a supplementary constraint on the number of activated resources k. Indeed the hysteresis property, for a single activation setting, insures that $q(., k)$ is decreasing in k. As an example : suppose that $q(m, 1) = 0$, then $q(m, 2) \leq 0$ which impose to test only two actions over three.

To integrate these properties, the policy improvement step is modified. We divide the process by the dimensions of the state (m, k). We first fix the level k and take the best actions for all m in an increasing order such that we consider the increasing property of q in m. For the hysteresis constraint, when we move from level k to level

$k+1$ we constraint the decisions set according to the decreasing property of q in k.

Algorithm 19: Policy Iteration with double level properties pseudo-code

Input: V^0, q^0
Output: q^*
Data: $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$ is known
/* Policy Evaluation */
for $s \in \mathcal{S}$ **do**
  $V^{t+1}(s) = \mathcal{R}(s, q^t(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, q^t(s)) V^t(s')$
/* Policy Improvement */
for $s \in \mathcal{S}$ **do**
  $q^{t+1}(s) = \text{argmax}_{a'} \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) V^t(s')$ with a' s.t. Double Level is satisfied between q^{t+1} and q^t
Until $q^t(s) = q^{t+1}(s)$

Algorithm 20: Policy Iteration with hysteresis properties pseudo-code

Input: V^0, q^0
Output: q^*
Data: $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$ is known
/* Policy Evaluation */
for $s \in \mathcal{S}$ **do**
  $V^{t+1}(s) = \mathcal{R}(s, q^t(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, q^t(s)) V^t(s')$
/* Policy Improvement */
for $s \in \mathcal{S}$ **do**
  $q^{t+1}(s) = \text{argmax}_{a'} \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) V^t(s')$ with a' s.t. Hysteresis is satisfied between q^{t+1} and q^t
Until $q^t(s) = q^{t+1}(s)$

On the other hand, we do not have any theoretical guarantee that these methods converge, first because we do not theoretically know if hysteresis policies are optimal for our model, and second because the underlying PI also has no convergence guarantees in multichain MDPs. Nevertheless, with reasonable input values in the numerical experiments we made (see Section 4.8), all the optimal policies returned by classical algorithms have hysteresis properties. Furthermore, all MDP algorithms considered here (structured as well as classical) returned the same solution. This therefore underlines the interest of considering such hysteresis policies especially since the gain in running time is obvious as observed in the experiments. Note that we also explored special input values that led us to different conclusions and that are displayed in Section 4.9.

Computation of the hysteresis thresholds The non-structured MDPs (also called simple MDPs) do not assume any restrictions for their policy research. Thus, they return the optimal policy and so return a decision rule q^* which gives the optimal action to take but not the thresholds. Therefore, we need to test the hysteresis property and to compute the l and L hysteresis thresholds consistently with Definition 5. Let q^* be the optimal policy returned by the PI algorithm. We check if q^* is monotone, if not the optimal policy is not hysteresis. If so, we proceed as follows. We consider, for all k, the set $\{m \mid q^*(m,k) = 1 \text{ and } q^*(m-1,k) = 0\}$. If the set is empty then $L_{k+1} = \infty$, else if it is of size 1 then $k+1 = m$ otherwise when the size is larger than 2

there are multiple values for a threshold and the policy is not hysteresis. Also, we consider, for all k, the set $\{m \mid q^*(m+1,k) = 0 \text{ and } q^*(m,k) = -1\}$. If he set is empty then $l_{k+1} = 0$, if the size is 1 then $l_{k+1} = m$ and otherwise the policy is not hysteresis.

4.6.3 Theoretical comparison between the two approaches MC and MDP

Although they seem very similar these two models present some rather subtle theoretical differences with notable consequences. They are studied here while the numerical comparisons will be carried out in Section 4.8.

4.6.3.1 Number of activated resources

We will see now that the hysteresis Definition 6 from Lui [98] used in the deterministic approach have worse results and that it does not activate the same number of VM than SMDP. Recall that strictly isotone assumption is required to keep the size of the chain constant (see [63]).

Proposition 3 (Suboptimality of strictly isotones policies). *The minimum computed with strictly isotone policies (and deterministic model) is greater or equal than the minimum computed with structured SMDP which, in turn, is greater or equal to the one computed with single MDP. Furthermore, in MC model it is not possible to activate (resp. deactivate) only a subset of the VM.*

Proof. The Markov chain approach (Section 4.3.2), deals with strictly isotone policies (*i.e.* $0 = l_1 < l_2 < \ldots < l_K \leq B$ and $0 = L_1 < L_2 < \ldots < L_K \leq B$). On the other hand, the SMDP model either considers isotone policy (see Definition 4) or does not assume any structural property at all. Since the constraints are more and more relaxed then the set of strictly isotone policy (MC model) is smaller or equal than the one of SMDP with isotone policy assumption which is smaller or equal than the one of single SMDP.

For the second point, since, L_K can not be infinite, all the K servers should be activated. On the other hand, since inequalities are strict then $l_K \geq K$, and the only state in which there is only one active server is restricted to the level $k = 1$. □

During the numerical experiments in Section 4.8, we identify numerous cases in which strictly isotone policies are not optimal. This is mainly due to the phenomenon of non activation or non deactivation. Hence examples of non-optimality are found in two specific categories of the whole categories described below. We noticed that these categories depend on the relative values of the parameters (λ, μ) between them. However, we do not know how to precisely quantify their borders.

Definition 8. These categories are :

1. Medium arrival case : All *VMs* are turned On and turned Off in both approaches. This occurs when the system load is *medium*, i.e. the arrival rate λ is close to the service rate $k \times \mu$.
2. Low arrival case : All *VMs* are turned On and turned Off in (MC) while in (MDP) some *VMs* will never be activated. This occurs when the system load is *low*, i.e. the arrival rate λ is very small compared to the service rate $k \times \mu$.
3. High arrival case : All *VMs* are turned On and turned Off in (MC) while in (MDP) some *VMs* will never be deactivated. This occurs when the system load is *high*, i.e. the arrival rate λ is very large compared to the service rate $k \times \mu$.

The proposition 3 highlights another benefit of MDP approaches since they allow to size the exact number of machines to be activated (this is particularly true in the low arrival case). Hence, if in a policy resulting from a MDP there exits a k such that for all $l > k$ and for all $m \in \{1, \ldots, B\}$ we have $q(m,l) < 1$, then $K-k$ machines are not necessary. This is not true in MC heuristics, nevertheless, it is possible to adapt the previous heuristics to

find the optimal number of VM to be activated in low arrival case. This requires running the algorithms for each level k with $k \in \{1, \ldots, K\}$. Then take the level k which has the smallest cost says k^*. If $k^* = K$ all machines must be turned on in K, otherwise only k^* machines are important and the other ones useless. But this method is time consuming. This method could be adapted for high arrival cases.

4.6.3.2 Different temporal sequence of transitions

The last point to investigate is the difference in dynamical behavior between transitions. This difference is slight and has no effect on average costs, nevertheless it induces a difference on the values of the thresholds. In this system, there are two kinds of transitions : natural transitions which are due to events (departures or arrivals) and triggered transitions which are caused by the operator (activation or deactivation). These two transitions are instantaneous. Due to their intrinsic definitions, the models considered here do not observe the system at the same epochs. Hence, in the Markov chain the system is observed just before a natural transition while in the MDP the system is observed just after a natural transition. More formally, let us assume that x is the state before a natural transition : it is seen by the Markov chain. Then the transition occurs and the state changes instantaneously in x' that is the state seen by the MDP. The controller reacts and the triggered transition occurs, thus the system moves in x'' in which the system remains until the next event which will cause the next natural transition. Since state changes are instantaneous, this has no impact on costs. We display in Figures 4.9 and 4.10 the temporal behavior for the Markov chain and MDP approaches.

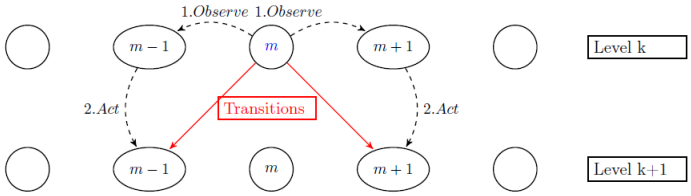


FIGURE 4.9 – Temporal behavior for the Markov chain approach

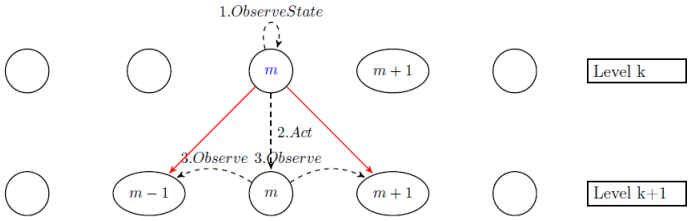


FIGURE 4.10 – Temporal behavior for the MDP approach

In the *medium case arrival* defined by Definition 8, thresholds are fully comparable and we have the following lemma :

Lemma 4. *In the medium case arrival, hysteresis thresholds of Markov chain and SMDP hysteresis threshold can be inferred from each other. Let* $(F_1, \ldots, F_{K-1})$ *and* $(R_1, \ldots, R_{K-1})$ *be the thresholds of the MC model and let* $(L_2, \ldots, L_K)$ *and* $(l_2, \ldots, l_K)$ *be the SMDP thresholds. Then, for all* $k \in \{1, \ldots, K-1\}$*, we have :* $L_{k+1} - 1 = F_k \quad$ *and* $\quad l_{k+1} + 1 = R_k$.

Proof. We make the proof for an activation threshold. Let assume that the state just before a transition is $x = (F_k, k)$, if an arrival event occurs then the system moves instantaneously in a state $x' = (F_k + 1, k)$.

According Definition 6, a virtual machine is activated and then the system instantaneously moves again in $x'' = (F_k+1, k+1)$. Observe now, that the state x' (according Definition 3) is the state in which the SMDP observes the system an takes its activation decision. Thus, we have $L_{k+1} = F_k + 1$. The proof works similarly for deactivation. □

4.7 Real model for a Cloud provider

We want to apply the algorithms presented before on concrete models derived from real environments values. We might now consider some true values of energy consumption, financial costs of virtual machines, and financial costs of Service Level Agreement. However, one of the difficulties of such an approach lies in the small number of works that consider both energy and quality of service. Hence, we should take different separate elements to build such a model. This allows us to assess the real impact, especially the financial cost and the reduction of the energy consumption, that a cloud owner can generate with such heuristics and to determine the optimisation method it should use.

4.7.1 Cost-Aware Model with Real Energy Consumption and Pricing

4.7.1.1 Energy Consumption Data

Many measurements are performed on the real datacenter *grid5000* in [25], and energy consumption data are obtained for VMs hosted on physical servers in [87]. We keep these values in Watt for energy consumption of virtual machines.

The work [25] details information about real energy consumption in virtualised system. The authors have made some simulations on a real Datacenter, *grid5000*, and have recovered datas about energy consumption for VMs based on some physical servers whom characteristics will be described in section 4.8.3. This will allow us to take these real values in Watt for energy consumption of virtual machines, when we built a concrete case.

4.7.1.2 Financial Cost

In order to keep the relevance of our cost-aware model, we must represent a financial cost. Henceforth, we transform energy consumption given in Watts into a financial cost based on the price in euros of the KWh in France fixed by national company. To obtain the financial values of the other prices, we observe the commercial offers of providers [13]. This gives us the operational costs.

Recall that the model proposed here is cost-aware and evaluates a financial cost for the cloud owner and that our goal was to minimise the operational cost while guaranteeing the QoS. One way to build real experiments is to consider real datas about energy consumption and to transform Watts consumption in a financial cost considering the price in euros of the KWh in France defined by EDF company. So, once we have our energy costs fixed depending on the system settings (CPU, vCPU, etc.), we can translate these watt costs into financial costs.

4.7.1.3 Service Level Agreement

We propose to modify the performance part of our cost function for capturing the realistic scheme of the Service Level Agreement. Actually, we have to give a concrete meaning to our holding cost since it does not translate directly from usual SLA models. Indeed, in SLA contracts, it could be stated that when the response time exceeds a pre-defined threshold T_{SLA} then penalty costs must be paid. Here, the penalty C_p is the price that a customer pays for one hour of service of a virtual machine (full reimbursement) [13] and the threshold corresponds to a maximal time to process a request.

With our model, it is possible to define an holding cost that models the penalty to pay when the QoS is not satisfied. We focus on the mean response time in the system where the response time is denoted by R_T. The SLA condition translates in : $E[R_T] \leq T_{SLA}$. We introduce the mean number of customers $E(N)$ computed with Little's law and assuming that the loss probability δ is negligible, i.e. $E[N] = E[R].\lambda(1-\delta) \implies E[N] = E[R].\lambda$. Therefore we have : $E[N] \leq T_{SLA} \cdot \lambda$.

We thus may define a customer threshold N_{SLA} such that $N_{SLA} = T_{SLA} \cdot \lambda$. This customer threshold is the maximum number of requests, that the system can accept to keep the required *QoS* satisfied. Therefore, each time $m > N_{SLA}$, the cloud provider will pay a holding cost penalty of C_p per customer.

In order to capture the realistic scheme of the Service Level Agreement, we modify our cost function by changing the performance part. Therefore the mean global cost for the cloud provider is :

$$\begin{aligned}\overline{C}(m,k) = & C_p.(m - N_{SLA}, 0)^+ + C_S.k \\ & + C_p.\lambda.\mathbb{1}_{\{m=B,k=K\}} + C_A.\lambda.\mathbb{1}_{\{m=F_k,\, k<K\}} \\ & + C_D.\mu.\min\{m,k\}.\mathbb{1}_{\{m=R_{k-1}+1,\, 2\leq k\leq K\}} \\ & + C_{static}\,,\end{aligned}$$

where C_{static} represents the static financial cost obtained from the idle energy consumption in Watt of the physical server hosting the virtual machines. The energy part of the cost function remains the same with the activation, deactivation, and energy cost of a VM being used. For the performance part, the issue we had was to give a meaning and real values for the holding cost since this one does not translate directly from usual SLA models. However, the holding cost can represent the penalty to pay if the QoS is not satisfied.

4.7.2 Real Packets traces and CPU utilisation

For obtaining concrete values of parameters λ and μ we must search workload traces with real scenarios in public cloud. Here the traces come from MetaCentrum Czech National Grid data [54].

After defining some real financial costs for energy and performance, we may want to give a meaning and real values to λ and μ. We consider in our model one request arriving at a time and one VM is processing one request at a time. Describing these parameters in this way, allows us to take some concrete values that can happen in real schemes. Looking at workload traces from MetaCentrum Czech National Grid datas [54], we can take a number of requests per hour arriving in a real data center. We also have some experimental results about the mean run-time of jobs, which give us real values of μ. Let us refer to 4.8.3 for numerical results with real scenarios.

Parameter values used in experiments can be found in Section 4.8.3.

4.8 Experimental Results

This part is devoted to numerical experiments and the comparison between all previous algorithms. We have implemented all methods in C++, with the stand-alone library MarmoteCore [66]. The experiments were launched with a processor Intel Xeon X7460 @2,66 GHz with 6x4 cores (24 threads) and 16 GB of RAM. The parameters of the system have been taken arbitrarily in a first phase in order to rank the different algorithms by comparing them to an exhaustive search when it is possible or to the one which has the best cost otherwise. We define four cloud scenarios that will be used for comparison in both : comparison of heuristics and comparison of heuristics with MDP approach. We call **Scenario A** a cloud system with $K = 3$ VMs and $B = 20$ maximum customers in the system, **Scenario B** a cloud system with $K = 5$ VMs and $B = 40$, **Scenario C** a cloud system with $K = 8$ VMs and $B = 60$ and **Scenario D** a cloud system with $K = 16$ VMs and $B = 100$ regarding the best solution among the heuristics. Finally, we tested in 4.8.3 our best algorithms on real scenarios.

4.8.1 Generic experiments to rank the algorithms

We first compared the heuristics to sort them and select the best ones for comparison with MDP approach. The results are displayed in the first part of Table 4.2. We present below some results for cloud systems with arbitrarily parameters. We compare all the algorithms presented above in terms of running time and in terms of optimality. The algorithms have been tested on 46656 instances obtained by combination of sets of fixed values : (λ, μ, costs). Costs (C_a, C_d, C_h, C_s) were taking values in $[0.5, 1, 2, 5, 10, 20]$, C_r in $[1, 10, 100, 1000, 5000, 10000]$, queueing parameters λ, μ in $[0.5, 1, 2, 5, 10, 20]$.

For Scenario A, accuracy is computed based on optimal solution generated by exhaustive search. For higher scale scenarios (B,C,D), we compute accuracy by comparing solutions with the best one among all heuristics. The results are given in Table 4.2. The mean execution time is given in the first column while the efficiency of algorithms is given in the last column.

Figure 4.11 displays the accuracy and time execution of the heuristics for the four cloud scenarios.

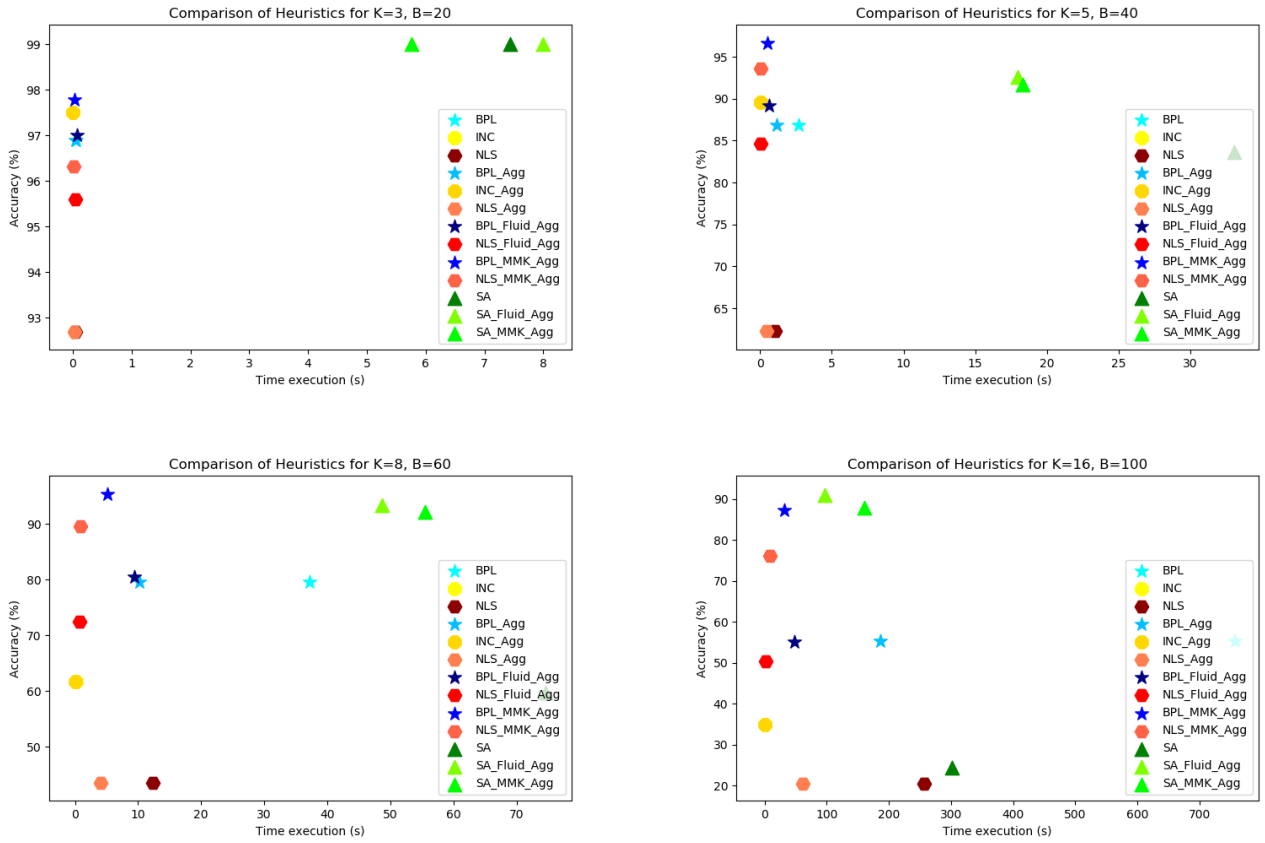


FIGURE 4.11 – Comparison of accuracy and time execution between heuristics

4.8.1.1 Analysis

Ranking of the heuristics Table 4.2 shows that the heuristics are very efficient for low scale systems such as $K = 3$ and $B = 20$. However, the efficiency of those algorithms is decreasing with the size of the system. This is due to the fact that the number of sets of thresholds tested by the heuristics compared to the total number of sets of thresholds decreases and therefore we potentially explore less candidates.

Looking at the results of the experiment, we detect that **BPL Agg** has the best percentage of optimality even for large systems. Unfortunately its running time is the highest. The fastest one, **INC Agg** returns the solution in less than a second but reveals an accuracy which is not the worse without being impressive. At last **NLS-Fluid Agg** presents a good compromise keeping a good precision and returning the solution in a short time.

Aggregation method strongly improves algorithms Table 4.2 show a significant time saving offered by the aggregation method for the same efficiency. In all cases, the aggregation method significantly improves our heuristics and the running times of all the algorithms are widely reduced. The improvement for three virtual machines is small, as expected, but the gain increases as the size of the system increases. Hence the running time is divided by about 1.3 when $K = 3$ and $B = 20$ and by about 3.8 when $K = 16$ and $B = 100$ (e.g. BPL algorithm passes from 757 seconds to 186 seconds).

Impact of the M/M/k approximation Coupled with heuristics **BPL** and **NLS**, $M/M/k$ approximation brings significant improvement considering the efficiency and the time execution. For a large scale case (Scenario D in Table 4.2), the *BPL MMK Agg* provides the best solution for 87.24 % of the instances in 31.7 seconds, which is the best heuristic in terms of time-accuracy ratio. We can notice, in all cases, that coupled heuristic always have better accuracy than the heuristic alone. Moreover, the initialisation technique does not only improve efficiency but also time execution. For example in Scenario C, **BPL Agg** has a mean time of 10,27 seconds for 79,57 % accuracy while **BPL-MMk Agg** obtains 95,39 % accuracy in 5,16 seconds, providing a double major gain.

Simulated annealing is not worth it We noticed that the simulated annealing suffers from a temporal overcost whatever the size of the system. Unfortunately this overcost does not allow to obtain a greater precision, since it obtains similar accuracy than some heuristics. Actually, it seems that greater precision could be obtained at the price of a very large running time.

4.8.2 Heuristics vs MDP

We compare next the best heuristics according to section 4.8.1 with the MDP algorithms. Numerical experiments have been done for the same Cloud model parameters and same system parameters. Results are displayed in Table 4.2. We first compare all the local search heuristics and their improvement in *MC Heuristics*. Then we compare all the (MDP) algorithms with the value iteration solution (convergence is known from Section 4.6.2.2) in *MDP*, before comparison with the heuristics to compare both approaches in *Comparison*. Again, we compare all methods for the four cloud scenarios A, B, C and D. For the heuristics, values in blue correspond to the minimum execution time for a given Scenario, values in red correspond to the best efficiency and values in green the ones that have the best ratio optimality-time. For the comparison between MDP and Heuristics, we only point out in green the methods with the best ratio.

4.8.2.1 Analysis

By Section 4.6.2.2, we theoretically know that value iteration algorithm converges to the exact solution. Therefore, we can use it as a benchmark to compare the MDP algorithms. Recall that we had no guarantee for convergence of unichain policy iteration algorithm. Yet, we observe in these numerical experiments, that all MDP algorithms obtain 100% of accuracy (optimal solution being given by value iteration) in all scenarios. Thus, all MDP algorithms converge to the optimal solution. Concerning the running time, we observe that policy iteration algorithms **PI** and **PI Adapted** are twice as good than value iteration on all studied scenarios. Furthermore, we see that the execution time of the structured MDP algorithms **DL-PI** and **Hy-PI** is divided by 2 compared to **PI**

Models	Algorithms	Scenario A		Scenario B		Scenario C		Scenario D	
		Time (sec)	% Opt	Time (sec)	% Min	Time (sec)	% Min	Time (sec)	% Min
	Exhaustive Search	4,16 sec	100 %	ND	ND	ND	ND	ND	ND
	Fluid approximation	0,00045 sec	47 %	0,002 sec	38,51%	0,012 sec	33 %	0,047 sec	21,7 %
	MMK approximation	0,001 sec	48 %	0,003 sec	38,87 %	0,029 sec	34 %	0,11 sec	15,37 %
	BPL	0,063 sec	96,9 %	2,684 sec	86,81%	37,23 sec	79,57 %	757 sec	55,39%
	BPL Agg	0,047 sec	96,9 %	1,163 sec	86,81%	10,27 sec	79,57 %	186 sec	55,39%
	BPL-Fluid Agg	0,07 sec	97 %	0,635 sec	89,13 %	9,35 sec	80,53 %	48 sec	55,05 %
	BPL-MMK Agg	0,036 sec	97,78 %	0,53 sec	96,59 %	5,16 sec	95,39 %	31,7 sec	87,24 %
MC Heuristics	**INC**	0,007 sec	97,51 %	0,018 sec	89,53%	0.157 sec	61,63 %	0.213 sec	34,95 %
	INC Agg	0,006 sec	97,51 %	0,009 sec	89,53%	0,048 sec	61,63 %	0,034 sec	34,95 %
	NLS	0.043 sec	92,70 %	1,007 sec	62,24 %	12,38 sec	43,51 %	257 sec	20,44 %
	NLS Agg	0,034 sec	92,70 %	0,458 sec	62,24 %	4,05 sec	43,51 %	61 sec	20,44 %
	NLS-Fluid Agg	0,039 sec	95,6 %	0,043 sec	84,60%	0,751 sec	72,39 %	2,03 sec	50,30 %
	NLS-MMK Agg	0,014 sec	96,32 %	0,06 sec	93,58 %	0,9 sec	89,54 %	8,39 sec	76,17 %
	SA	7,43 sec	99 %	33,1 sec	83,62 %	74,59 sec	59,68 %	302 sec	24,37 %
	SA-MMK Agg	5,76 sec	99 %	18,32 sec	91,73 %	55,44 sec	92,13 %	245 sec	90,96 %
	SA-Fluid Agg	8 sec	99%	17,98 sec	92,57%	48,7 sec	93,38 %	160 sec	87,78 %
		Time (sec)	% Opt	Time (sec)	% Opt	Time (sec)	% Opt	Time (sec)	% Opt
	VI	0.0057 sec	100 %	0.021 sec	100 %	0.0406 sec	100 %	0,0944 sec	100 %
	RVI	0.0057 sec	100 %	0.021 sec	100 %	0.0406 sec	100 %	0,095 sec	100 %
MDP	**PI**	0.0028 sec	100 %	0.0124 sec	100 %	0.0214 sec	100 %	0.0606 sec	100 %
	PI Adapted	0,0023 sec	100 %	0,0117 sec	100 %	0,0201 sec	100 %	0,0583 sec	100 %
	DL-PI	0,00115 sec	100 %	0,0072 sec	100 %	0,0105 sec	100 %	0,0452 sec	100 %
	Hy-PI	0,00113 sec	100 %	0,0069 sec	100 %	0,0100 sec	100 %	0,0442 sec	100 %
		Time (sec)	% Opt	Time (sec)	% Opt	Time (sec)	% Opt	Time (sec)	% Opt
	NLS-MMK Agg	0,014 sec	64,82 %	0,06 sec	61,52 %	0,9 sec	59,82 %	8,39 sec	52,61 %
Comparison	**BPL-MMK Agg**	0,036 sec	65,15 %	0,53 sec	62,65 %	5,16 sec	61,69 %	31,7 sec	57,1 %
	VI	0.0057 sec	100 %	0.021 sec	100 %	0.0406 sec	100 %	0,0944 sec	100 %
	DL-PI	0,00115 sec	100 %	0,0072 sec	100 %	0,0105 sec	100 %	0,0452 sec	100 %
	Hy-PI	0,00113 sec	100 %	0,0069 sec	100 %	0,0100 sec	100 %	0,0442 sec	100 %

TABLE 4.2 – Numerical experiments for comparison of heuristics and MDP algorithm

Adapted. This leaves us to conclude that integration of policy with structural properties strongly accelerates the convergence.

4.8.2.2 Comparison between (MDP) and (MC) approaches

We now compare which approach works best to obtain optimal thresholds. We first observe that the accuracy difference is striking, indeed, the MC approach is optimal only about half the time. Indeed, from Section 4.6.3.1, we know that depending on the queueing parameters, (MDP) approach could not activate or deactivate certain virtual resources whereas the (MC) approach always find thresholds for all levels due to constraints on the searched policy. This is why (MC) heuristics are outperformed by (MDP) algorithms. Hence, for many instances their optimal cost is higher due to additional activation or deactivation thresholds, generating extra costs. We see that MDP algorithms are faster than heuristics in the four scenarios. Hence for small scenarios **Hy-PI** is about 100 times faster than the best heuristic and about 200 times faster for large scenarios. Therefore, the MDP approach is significantly better than the (MC) approach.

4.8.2.3 High-scale simulations MDP

We ran numerical simulations for high-scale scenarios (real size datacenter) with best MDP algorithm **Hy-PI** to assess its performance. Simulations were performed on pre-selected costs and queuing parameters. Our sizes K and B are larger than these in previous works e.g. [103]. Results are displayed in Table 4.3. We also display the state space size and the results of **BPL-MMK Agg** to show that they (heuristics) do not scale. Observe that, for small data centers (64 VM and 400 customers) the optimal policy can be computed in a reasonable time in practice : 2 seconds. However, for large data centers some further researches should be done since the time represents around 6 hours for a scenario with 1024 VMs and 6400 customers. We fill the table with ND (Not defined) for the heuristic when the execution time is above 10 hours. We show that the best heuristic already suffers in the case of 64 VM and 400 customers therefore demonstrating that heuristics do not scale to large Cloud systems.

	K=3, B=20	K=5, B=40	K=8, B=60	K=16, B=100	K=32, B=200	K=64, B=400	K=128, B=800	K=256, B=1600	K=512, B=3200	K=1024, B=6400
State Space	60	200	480	1600	6400	25600	102400	409600	$1,6 \cdot 10^6$	$6,5 \cdot 10^6$
Hy-PI	0.0002	0.0019	0.0274	0.0517	0.3023	2,4238	37,97	329.54	2327	6.3 H
BPL-MMK Agg	0.036	0.53	5.16	31.7	590	ND	ND	ND	ND	ND

TABLE 4.3 – High-scale datacenter resolution with MDP algorithm **Hy-PI**

4.8.3 Numerical experiments for concrete scenarios

Data and scenarios We define now numerical values to the real model of Section 4.7. Several case-studies as well as several performance and metrics curves are used to select the values. Throughout this section the time unit is the hour. The Cloud platform studies come from [87] : the physical hardware is a Taurus Dell PowerEdge R720 server having a processor Intel Xeon e5-2630 with 6×2 cores up to 2.3GHz. The processor hosts virtual machines by an hypervisor. One core is considered to be one vCPU and cores are distributed equally among the virtual machines : 12 VMs with 1 core per VM, 6 VMs with 2 cores per VM, and 3 VMs with 4 cores per VM. These three different cases are selected such that we can use the prices of Amazon EC2 instances [13].

Energy consumption values come from [87]. Physical hardware consumption is around 100W. The dynamic consumption of the virtual machines is stated in accordance with the number of cores per VM. Namely, the Watt consumption for activation, deactivation and use is different as it depends on the characteristics of the virtual

TABLE 4.4 – Parameter values for three scenarios

Parameter	Model A	Model B	Model C
$\#VMs$	3	6	12
Instances	a1.xlarge	t3.small	t2.micro
$\#vCPUs$	4	2	1
RAM(Go)	8	2	1
C_p	0.0914€/h	0.0211€/h	0.0118€/h
C_S	0.00632€/h	0.00316€/h	0.00158€/h
$C_A = C_D$	0.00158€	0.00079€	0.00032€
C_{Static}	0.0158€/h	0.0158€/h	0.0158€/h
B	100	100	100
λ	50 req/h	50 req/h	50 req/h
μ	20 req/h	10 req/h	5 req/h

machines hosted on the physical server. Hence, 1 VM with 4 vCPU which executes a request requires 40W to work while its activation and deactivation take a power of 10W. At last, all the energy consumption values are transformed in financial costs.

Some workload samples which come from real traces of the MetaCentrum Czech national grid are detailed in [54]. We select some samples to build concrete daily scenarios for huge size requests. Hence, we pick a number of arrivals per hour of $\lambda = 50$ and number of served requests per hour as $\mu = 5, 10$ or 20. Table 4.4 details all the parameter values for each model.

Next, we focus on three concrete cases depending on the values chosen for the SLA. We give for each of them, the optimal cost and the optimal policy considering each model (A,B,C) that have been calculated with the best algorithm **Hy-PI**.

The results for $N_{SLA} = 10$ are :

1. Model C :
   - mean hourly cost = 0,0778 €/h
   - optimal policy : F=[1,2,4,5,7,8,9,10,11,12] ; R=[0,1,2,3,4,5,6,7,8,9,10].
2. Model B :
   - mean hourly cost = 0.0798 €/h
   - optimal policy : F=[1,3,4,5,7] ; R=[0,1,2,3,4].
3. Model A :
   - mean hourly cost = 0.136 €/h
   - optimal policy : F=[1,3] ; R=[0,1].

The results for $N_{SLA} = 30$ are :

1. Model C :
   - mean hourly cost = 0.0354 €/h
   - optimal policy : F=[2,4,5,7,8,10,12,14,16,18,19] ; R=[0,1,2,3,4,5,6,7,8,9,10].
2. Model B :
   - mean hourly cost = 0.0358 €/h
   - optimal policy : F=[3,6,9,12,16] ; R=[0,1,2,3,6].
3. Model A :

* mean hourly cost = 0.0397 €/h
* optimal policy : F=[5,10] ; R=[0,1].

The results for $N_{SLA} = 50$ are :

1. Model C :
   * mean hourly cost = 0,0324 €/h
   * optimal policy : F=[3,5,7,9,11,14,17,21,26,30,34] ; R=[0,1,2,3,4,5,6,7,10,14,19].
2. Model B :
   * mean hourly cost = 0,0326 €/h
   * optimal policy : F=[4,8,13,21,30] ; R=[0,1,2,5,14].
3. Model A :
   * mean hourly cost = 0,0334 €/h
   * optimal policy : F=[9,22] ; R=[0,5].

We provide the mean run time of the best algorithm (**Hy-PI**) among concrete experiments : 0,00127 s for Model A ; 0,0073 sec for Model B and 0,0372 sec for Model C. The optimal solution is computed in a short-time and therefore can allow to recompute new policy when the demand is evolving, in order to dynamically adapt to the need.

Mean global costs given SLA parameter or arrival rate From numerical simulations it could be noticed first that, for a fixed arrival rate λ, when the SLA is not so strict, then the cost decreases. Second that, when the response time set by the SLA rises, then the operating costs for the cloud providers are reduced. Indeed, providers will pay fewer penalties to clients and fewer power bills for running VMs. On the other hand, we observe that Model C is always better than Model B and Model A for any possible values of N_{SLA} and for any values of λ given a fixed SLA parameter. We can conclude that it is always better to decompose physical machine in several virtual resources, since it allows a more efficient resource allocation in dynamic scenario. Indeed, the more virtual resources you have, the less you will pay per utilisation since they require less CPU consumption. Moreover, it allows a more flexible system where you can easily adapt virtual resources to the demand.

Thresholds given SLA parameter or arrival rate We also observe that Model C always activates a second virtual resource before the two other models B and A, for any fixed values of λ and SLA parameter N_{SLA}. Furthermore, when the arrival rate λ is fixed, then all models will activate later when the SLA is less restrictive. Lastly, when the SLA parameter is fixed, we notice that the first activation threshold decreases when λ increases. Indeed, if the demand is growing, it requires more resources and requires faster activation of virtual resources.

Cost evaluation We display on Figure 4.12 an example ($K = 8, B = 60$) which exhibits the evolution of the financial cost per hour according to a given N_{SLA}, or to the load. We can see in the figure the impacts of such parameters regarding performance and energy costs.

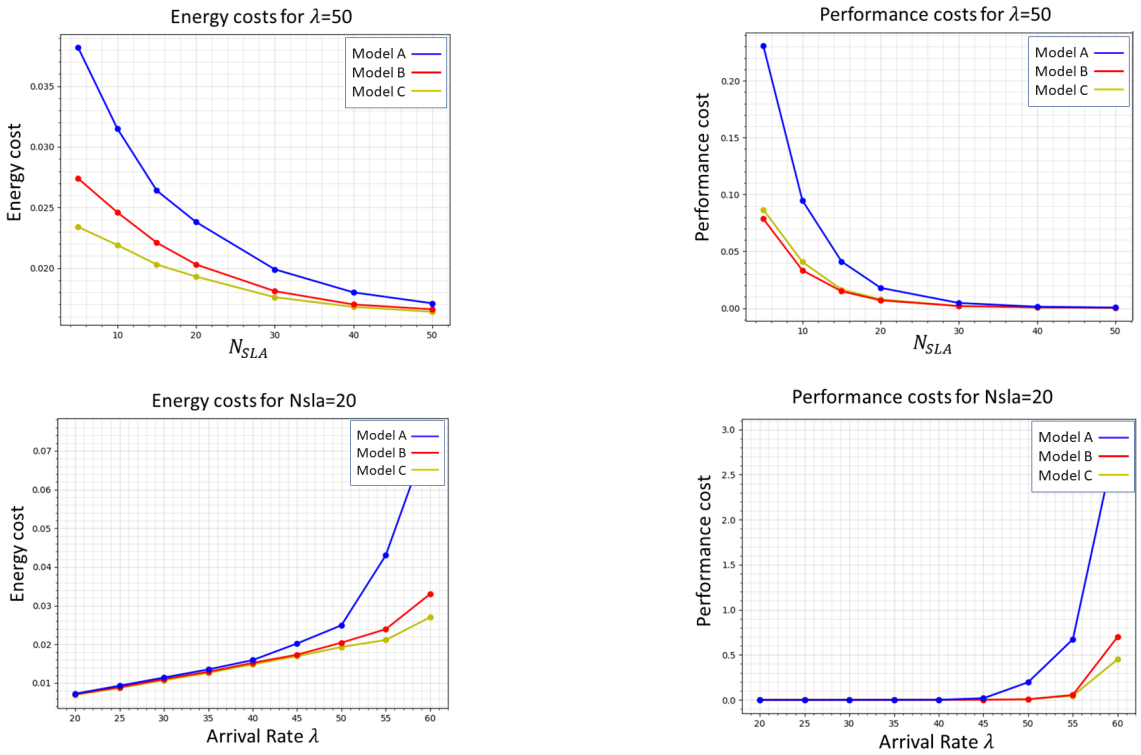


FIGURE 4.12 – Energy and performance costs given fixed SLA or arrival rate

Metrics and Performance We want to give some performance behaviour depending on some parameters, mostly how the financial cost per hour evolves depending on the given N_{SLA}, or on the load. The cost is calculated with the BPL Agg heuristic for these examples. We can see in Fig. 4.13 the impact of the SLA on the financial hourly cost for a cloud provider.

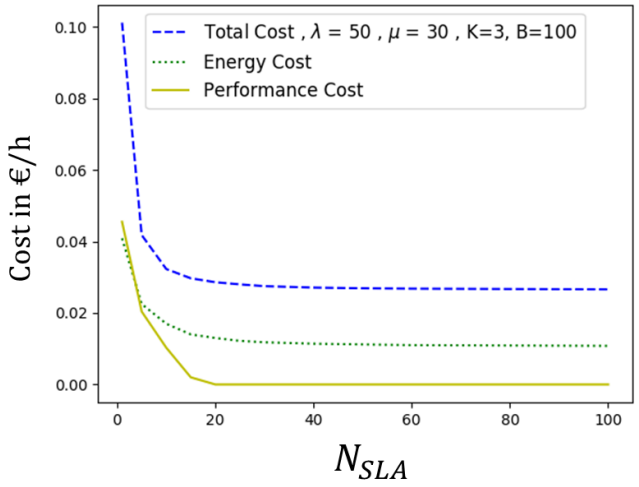


FIGURE 4.13 – Financial Cost for an hour depending on the SLA

Fig. 4.14 shows the evolution of the cost depending on the workload. It has been drawn with the parameters of Model B and a fixed SLA : $N_{SLA} = 30$.

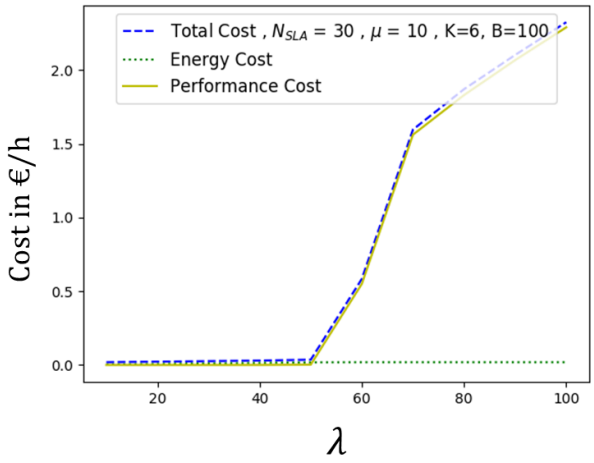


FIGURE 4.14 – Financial Cost for an hour depending on the workload

4.9 Are hysteresis policies really optimal ?

This section relies on structural properties of the optimal policy and provides a counter-example to contradict usual statements about optimality of hysteresis policies. Several works have tried to prove the optimality of special kind of policy (stationary, threshold, hysteresis) in this type of cloud models. The different works deal with optimal control for dynamic admission of requests in a queue and the service rate control to allocate in the queue depending on the number of requests. As an example, Yang et al. [151] proved that optimal policy is of hysteresis type in a similar cloud model. They study a scenario where a data center has multiple data servers to deal with jobs and the servers are switched into sleeping mode in periods of low traffic load to reduce energy consumption. The authors formulate the problem as a MDP and show that the optimal policy has a double threshold structure. Yet their cost function is different from our model since they do not consider activation, deactivations costs. Moreover, Szarkowicz et al [134] study the problem of determining optimal operating policies for an M/M/S queuing system. Their system state is also defined by the number of customers in the system, and the number of active service channels. Their cost structure includes customer holding and service channel operating costs as well as a linear switching cost. In their work, they also show the optimality of hysteresis policies. Last, Maccio and Down [99] state that for all energy-aware systems and for all linear well-formed cost functions, the optimal policy is a threshold policy. In addition, they emphasise that in general these policies are hysteretic in nature due to separate thresholds.

All numerical experiments provided in section 4.8 turned out to return hysteresis policies when calculation were made by MDP algorithms (e.g. VI or Hyst-PI). However we have remarked when manipulating the Cloud parameters that it could exist optimal policies returned from VI (with proven convergence) that were not of hysteresis type. We provide here a counter-example in a particular cloud system where the optimal policy returned by MDP algorithm is not of hysteresis type. Notice that this example relies on cloud parameters that do not make sense in real cloud environments but still shows mathematically that we can not prove optimality of hysteresis structure here, since it will not be true for all cloud environments. In contrast to the works of [78], [58] showing

optimality of hysteresis policies for their queuing model, this is not true in our scenario due to the addition of activation, deactivation and reject costs.

Example 3. *Let us define the following cloud environment with :*
$K = 3$, $B = 10$
$C_A, C_D, C_R, C_H, C_S = 50000, 0.05, 1000, 1, 0.05$
$\lambda, \mu = 20, 5$
The optimal policy returned by the Value Iteration algorithm, which has proven convergence is displayed in Table 4.5. Notice that the Policy Iteration algorithm has returned the same policy and cost.

Policy	**Number of jobs** m	$k=1$	$k=2$	$k=3$
$q(m,k)$	0	$q(m,k) = -1$	$q(m,k) = 0$	$q(m,k) = 0$
	1	$\boxed{q(m,k) = -1}$	$\boxed{q(m,k) = 0}$	$q(m,k) = 0$
	2	$q(m,k) = 1$	$q(m,k) = 0$	$q(m,k) = 0$
	3	$q(m,k) = 1$	$q(m,k) = 1$	$q(m,k) = 0$
	4	$q(m,k) = 1$	$q(m,k) = 1$	$q(m,k) = 0$
	5	$q(m,k) = 1$	$q(m,k) = 1$	$q(m,k) = 0$
	6	$q(m,k) = 1$	$q(m,k) = 1$	$q(m,k) = 0$
	7	$q(m,k) = 1$	$q(m,k) = 1$	$q(m,k) = 0$
	8	$q(m,k) = 1$	$q(m,k) = 1$	$q(m,k) = 0$
	9	$q(m,k) = 1$	$q(m,k) = 1$	$q(m,k) = 0$
	10	$q(m,k) = 1$	$q(m,k) = 1$	$q(m,k) = 0$

TABLE 4.5 – Policy $q(m,k)$ returned by Value Iteration showing non-hysteresis

We recall that the hysteresis policy should satisfy two properties :

* $q(.,k)$ *increases*
* $q(m,.)$ *decreases*

In this example, the first property is satisfied since we always have for a given $k : \forall m \;\; q(m,k) \leq q(m+1,k)$. *Yet the second property is violated since we have :* $q(1,1) = -1$ *and* $q(1,2) = 0$ *which contradict the hysteresis property of* $q(m,.)$ *being decreasing. This is due to cost choices that are drastically different, which does not provide realistic meaning but prevents us to show optimality of hysteresis policies in our model, for all cloud environments.*

The intuition here is that when we take a very high activation cost C_A *compared to the others, the agent does not chose to activate in states where the load is very low such as* $m = 0$ *or* $m = 1$..

4.10 Discussion

4.10.1 Summary of the chapter

In this chapter we have compared theoretically and numerically two different approaches to minimise the global cost and to find auto-scaling thresholds policies integrating both performance and energy consumption in an auto-scaling cloud system when the agent is provided the full model of the cloud environment. The relevance of this study for a cloud provider is to provide an auto scaling resource allocation policy very quickly to minimise its financial cost (around 38 seconds for 128 VMs and 800 customers with the best MDP algorithm). We exhibit that the best static heuristics are strongly outperformed by SMDP models and that hysteresis presents a significant

improvement. Although we have provided a counter-example with specific costs parameters for optimality of hysteresis policies, it turns out that the optimal policies returned from numerical experiments were all hysteresis when taking consistent costs. From the numerical results, we show that the (MDP) approach is faster, but also much more accurate. Moreover, it allows in some cases to define the number of resources needed to be allocated on the physical server. Finally, we devised a realistic scenario of our cloud model we assessed our best methods and analysed numerical results depending on SLA parameters and queueing parameters.

4.10.2 How to proceed with unknown statistics

This chapter assumes that the autonomous agent fully knows the cloud environment and its dynamics, which allows him to compute optimal policies. In real cloud systems, this hypothesis no longer holds since it is difficult to capture queuing statistics such as arrival rates, service rates, etc. In this context we need to consider RL techniques that can learn in unknown environments.

CHAPTER 5

MODEL-BASED REINFORCEMENT LEARNING FOR MULTI-TIER NETWORK

The contributions of this chapter have been published in [142] :

* Thomas Tournaire, Jeanne Barthélémy, Hind Castel-Taleb, Emmanuel Hyon. Reinforcement Learning with Model-Based Approaches for Dynamic Resource Allocation in a Tandem Queue. In *Performance Engineering and Stochastic Modeling. Springer International Publishing (ASMTA 21)*, 2021

The challenge for resource allocation in Cloud networks is that human experts do not easily know the exact statistics of the system (arrivals, services) in real environments, because they evolve too quickly. Therefore a software agent does not possess any environment model for computation of the optimal policy. This requires the implementation of Reinforcement Learning solutions. On the other hand, we are not sure that the practitioners approaches are robust (often very experimental). Most of the work concerning RL for Cloud resource allocation is dedicated to model-free techniques such as Q-Learning [39] or Deep RL [113], but there exists no application of model-based techniques for such use-cases.

In this chapter, we want to evaluate the model-based RL approaches in the case of queuing networks, which has been little done so far. Moreover, we want to extend the range of models studied (in Chapter IV to go beyond the simple single multi-server model) to networks of several queues, which is better able to represent multi-tier architectures, which has also not been done much in the literature. Consequently, we now assume that the learning agent does not carry the knowledge of the MDP model. The aim is therefore to adopt model-based RL approaches [106], especially Dyna architectures [8] and MDP online techniques [27] to assess their relevance, if ever, or to improve them for dynamic resource allocation in queuing network systems. Moreover, we want to assess the robustness of such approaches in use-cases where there are arrivals that are bursty.

Therefore, the key contributions of this chapter are as follows :

* We extend the one node scenario and propose two queuing systems for modelling three-tier architectures : tandem queueing model with Poisson arrivals, and then with (MMPP) Markov modulated Poisson process ;
* We integrate and discuss literature considerations regarding experimental comparison of reinforcement learning algorithms ;
* We implement several versions of Dyna architecture and experimentally show that model-based reinforcement learning techniques can outperform classical model-free algorithms such as Q-Learning, on cloud auto-scaling applications ;

* We study the robustness of RL algorithms on a partially observable system with MMPP arrivals where arrival rate is varying over time and is hidden to the agent. Nevertheless, we show that deterministic model-based methods suffer more than Q-learning in changing environments, while stochastic model-based methods such as MDP online can manage to have better performances.

The chapter first gives a more detailed overview of model-based reinforcement learning techniques, next presents two queuing models to express multi-tier network architectures. Then it shows how we have selected model-based RL techniques by providing a general algorithm and how we parameterised them to compare with state of the art model-free RL on the multi-tier cloud network. Last, we present comparison results between RL techniques on different cloud scenarios and parameters in a simulated environment, demonstrating the gain of model-based RL techniques.

5.1 Background Model-Based Reinforcement Learning

Now, we present in more details the model-based RL framework introduced in Chapter III. We first describe Dyna architectures [8] and prioritised replay techniques [122], state of the art MDP online techniques [27], [74], [109] and more advanced techniques following [106]. More precisely, we describe in this background section how the RL agent can learn the model of the world and how it can integrate learning and planning. We encourage readers to check out the section 3.2.4.2 in Chapter III that discusses the planning phase and especially how it is performed : Simulated trajectories, breadth and depth, dynamic programming. Broadly, the planning phase consists of simulating trajectories with the learned model for updating the value or policy functions and the learning phase consists of extracting an environment model with dynamics and reward functions from collected data. Next, we provide more details about SoTA model-based RL algorithms regarding their learning and planning processes.

5.1.1 Dyna architectures - Deterministic models and buffer replay

Dyna architectures [8] aims to improve model-free methods by adding a supplementary planning phase where the agent can update more often its state-value function from experiences. The collected samples $(s^t, a^t, r^{t+1}, s^{t+1})$ can be used to directly improve the value function and policy by storing them into a *buffer replay*. This buffer replay serves as a *deterministic* model of the world by predicting what will be the reward r^{t+1} and the next state s^{t+1} when the agent selects action a^t in state s^t. The interaction between experience, model, values, and policy are depicted in Figure 5.1. The experience can improve value and policy functions either directly or indirectly via the model.

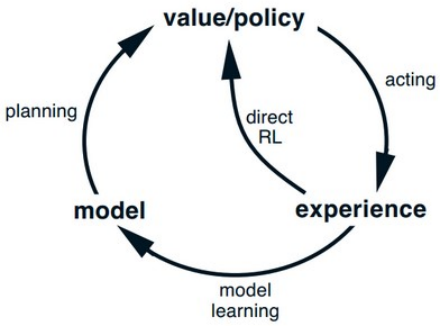


FIGURE 5.1 – The reinforcement learning scheme in Dyna architectures (Figure from [8])

5.1.1.1 Dyna-Q

Dyna-Q [133] is the natural Dyna architecture built upon the model-free *Q-Learning* algorithm. In this algorithm, the learning agent continues to update its Q-value function with the one-step Q-learning update equation $(Q(s,a) \leftarrow Q(s,a) + \alpha(r + \max'_a Q(s',a') - Q(s,a))$ in a model-free manner. However, it also stores the collected experiences obtained from environment interactions in a buffer that is reused for planning : $\mathcal{D}_i = \{< s,a,r,s' >_1, < s,a,r,s' >_2, \cdots, < s,a,r,s' >_i\}$. The planning method is the random-sample one-step tabular Q-planning method, i.e. that the agent randomly select a state-action pair (s,a) that have been experienced and generates the outcome (s',r) from the buffer model to re-update its Q-value function.

In more detail, after each transition $s^t, a^t r^{t+1}, s^{t+1}$, the model records in its table entry for s^t and a^t the prediction that r^{t+1}, s^{t+1} will deterministically follow. The latter is referred as *model-learning*. Next, it does a planning phase by selecting randomly from the buffer a state–action pair that has been experienced before, and simply returns the last-observed next state and next reward as its prediction. This step is actually a one-step simulation used for planning and update of the Q function.

In summary, the Dyna-Q agent alternatively updates its Q-value function for a pair (s,a) either with the Q-Learning equation, from direct experiences (model-free) or from the planning with buffer replay (model-based). Conceptually, planning, acting, model-learning, and model-free reinforcement learning occur simultaneously and in parallel in Dyna agents. We provide the pseudo-code of the Dyna-Q learning method in Algorithm 21. We denote by $\mathcal{M}(s,a)$ the buffer replay model that predicts the outcome for a state-action pair (s,a), and by ϵ-greedy(Q,s) the policy that randomly exploit or explore.

Algorithm 21: Dyna-Q Algorithm [8]

Input: $Q_0(s,a)$, $\mathcal{M}(s,a)$ empty buffer, ϵ
Output: Q^*
Data: System dynamics $\mathcal{P}$ and reward $\mathcal{R}$ unknown

```
/* Loop until end of episodes                                          */
for e ∈ MaxEpisode do
    Select state s_0 ∈ S                                   // Initial state
    for i ∈ MaxIteration do
        a ← ε − greedy(Q, s)
        Execute action a and observe resultant reward r and new state s'
        Q(s,a) ← Q(s,a) + α[r + γmax'_a Q(s',a') − Q(s,a)]   // Model-free update
        M(s,a) ← r, s'                                       // Model-learning
        /* Planning                                                    */
        for j ∈ MaxPlanningSteps do
            s ← randomly selected from buffer
            a ← randomly selected from actions previously taken in s
            r, s' ← M(s,a)                                   // Simulating
            Q(s,a) ← Q(s,a) + α[r + γmax'_a Q(s',a') − Q(s,a)]   // Planning update
```

5.1.1.2 Dyna-Q-+

Deterministic models learned by the agent in Dyna architectures might however be incorrect because the environment is stochastic and only a limited number of samples have been observed. Moreover, the environment might have changed and its new behavior has not yet been observed by the agent. When the model is poorly

approximated, the planning process will compute a suboptimal policy. This requires the agent to explore states where the dynamics might have changed so it can adjust its predictive model for better planning. One heuristic developed by Sutton [8] is *Dyna-Q-+*, where the principle is to give incentive to the learning agent for exploration of state-action pairs that have not been seen for a given time. For this purpose, the agent keeps track for each state–action pair of how many time steps have elapsed since the pair was last tried in a real interaction with the environment. The more time that has elapsed, the greater the chance that the dynamics of this pair has changed and that the current model of it is incorrect. To encourage the phenomenon of visiting long-time unseen state-action pairs, we provide an *extra reward* κ (relatively small) so the agent will be attracted by these pairs. Formally, we denote $\tau_{s,a}$ the time steps passed after the agent visited the pair (s, a) and for which it got the reward r. In the buffer replay, the reward r is changed into $r + \kappa.\sqrt{\tau_{s,a}}$. The planning steps considering these pairs will therefore update the Q-value function in this direction. The process is summarised in Algorithm 22.

Algorithm 22: Dyna-Q-+ Algorithm [8]

Input: $Q_0(s, a)$, $\mathcal{M}(s, a)$ empty buffer, ϵ, κ, τ
Output: Q^*
Data: System dynamics $\mathcal{P}$ and reward $\mathcal{R}$ unknown
/* Loop until end of episodes */
for $e \in MaxEpisode$ **do**
  Select state $s_0 \in \mathcal{S}$ // Initial state
  for $i \in MaxIteration$ **do**
    $a \leftarrow \epsilon - greedy(Q, s)$
    Execute action a and observe resultant reward r and new state s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max'_a Q(s', a') - Q(s, a)]$ // Model-free update
    $\forall (s, a) \in Buffer : \tau_{s,a} \leftarrow \tau_{s,a} + 1$
    $r \leftarrow r + \kappa.\sqrt{\tau_{s,a}}$
    $\mathcal{M}(s, a) \leftarrow r, s'$ // Model-learning
    /* Planning */
    for $j \in MaxPlanningSteps$ **do**
      $s \leftarrow$ randomly selected from buffer
      $a \leftarrow$ randomly selected from actions previously taken in s
      $r, s' \leftarrow \mathcal{M}(s, a)$ // Simulating
      $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max'_a Q(s', a') - Q(s, a)]$ // Planning update

5.1.1.3 Prioritised replay techniques integrated in Dyna architecture

In the two Dyna-Q algorithms presented before, simulated transitions are started in state–action pairs selected uniformly at random from all previously experienced pairs. But a uniform selection is usually not the best and planning can be much more efficient if simulated transitions and backups focus on particular state-action pairs. Silver et al. [122] cover several techniques to prioritised the replay buffer for Deep-Q-network algorithm such that the Q neural network trains on pertinent samples. Fortunately, this can be used in the Dyna architectures. The goal is to replay in the planning phase the important transitions more frequently, and therefore learn more efficiently. The key idea is that an RL agent can learn more effectively from some transitions than from others. In particular, Silver et al. propose to more frequently replay transitions with high expected learning progress, as measured by the magnitude of their temporal-difference (TD) error.

Prior replay buffer with TD-error The first criterion to consider would be the amount the RL agent can learn from a transition in its current state (expected learning progress). While this measure is not directly accessible, a reasonable proxy is the size of a transition's TD error δ, which indicates how unexpected the transition is : specifically, how far the value is from its next-step bootstrap estimate. The algorithm *Dyna-Q-prior* stores in the buffer memory, in addition to the quadruple (s, a, r, s'), the TD error computed in the model-free update equation. The selection process for the planning phase is to select state-action pairs with the largest absolute TD error. Then, as the usual Dyna algorithm, a Q-learning update is applied to this transition, which updates the Q-function in proportion to the TD error. Obviously this adds complexity due to larger buffer memory but also the sorting process to select pairs with highest TD error. The process is summarised in Algorithm 23.

Algorithm 23: Dyna-Q-prior Algorithm [8]

Input: $Q_0(s, a), \mathcal{M}(s, a)$ empty buffer, ϵ
Output: Q^*
Data: System dynamics $\mathcal{P}$ and reward $\mathcal{R}$ unknown

```
/* Loop until end of episodes                                        */
for e ∈ MaxEpisode do
    Select state s_0 ∈ S                                  // Initial state
    for i ∈ MaxIteration do
        a ← ε − greedy(Q, s)
        Execute action a and observe resultant reward r and new state s'
        δ(s, a) = r + γ max'_a Q(s', a') − Q(s, a)
        Q(s, a) ← Q(s, a) + αδ(s, a)                  // Model-free update
        M(s, a) ← r, s', δ(s, a)                      // Model-learning
        /* Planning                                                  */
        for j ∈ MaxPlanningSteps do
            s, a ← with highest δ(s, a)
            r, s' ← M(s, a)                                // Simulating
            Q(s, a) ← Q(s, a) + αδ(s, a)              // Planning update
```

Stochastic prioritisation The principal issue of prioritising the replay buffer with the TD-error is that state-action pairs with low TD errors that have been experienced on first visit, might not be replayed for a long time in the planning process. To overcome this issue, the work [122] introduces a stochastic selection process that lies between pure greedy prioritisation on the TD-error and random sampling, by associating to each action-pair a probability $h(s, a) = \dfrac{1}{rank(s, a)}$ where $rank(s, a)$ is the position of the sample in the replay buffer, sorted by the highest TD-error. This gives a chance for experiences with low TD-error to be selected for planning.

5.1.2 Real Time Dynamic Programming (RTDP)

The major drawback with Dyna architectures is that they consider (or assume) a deterministic model of the world. In stochastic environment with changes, this becomes a major problem since the update of the value function is made with experiments that do not integrate the system's randomness. It exists techniques that aim to learn a stochastic model of the world, by learning the probability function $\mathcal{P}$.

5.1.2.1 The primary concept : adaptive control

In 1999, Barto et al. [23] have covered the field of Dynamic Programming extensively. In their paper, they study *Real Time Dynamic Programming* techniques where a learning agent updates its value function or policy with DP techniques and always decide to act greedy, i.e. selecting $a = \text{argmax}_a Q(s, a)$. In addition, they study what they call *adaptive control*, which is MDP scenario with incomplete information, i.e. where the agent does not know the dynamics of the system. Notice that this is in fact Reinforcement Learning. The authors claim that it exists two major classes of adaptive control methods for MDP with incomplete information : *Bayesian methods* and *non-Bayesian methods*. The former (Bayesian) assumes a known a priori probability distribution over the class of possible stochastic dynamic systems. As observations accumulate, this distribution is revised via *Bayes' rule*.

The latter (non-Bayesian) explicitly models the dynamic system being controlled and will be considered in this chapter. They use system identification algorithms to update parameters whose values determine the current system model. They typically make control decisions under the assumption that the current model is the true model of the system, i.e. they consider the confidence in their model used for planning. In a nutshell, these non-Bayesian methods estimate the unknown state-transition probabilities and immediate costs based on the history of state transitions and immediate costs observed while the controller interacts with the environment. The most common form of this identification approach is the maximum-likelihood technique [8], that allows to update the transition model by counting occurrences of visited state-action pairs (s, a) and (s, a, s') :

$$\mathcal{P}(s'|s, a) = \frac{\sharp(s, a, s')}{\sharp(s, a)}$$

Last, it appears in these methods a conflict between conducting enough exploration to achieve model convergence and the objective of eventually following an optimal policy. This is related to the *exploration-exploitation* dilemma of the RL field. Adaptive optimal control algorithms require mechanisms for resolving these problems, but no mechanism is universally favored.

5.1.2.2 PAC-MDP algorithms

In this section, we give an overview of model-based RL techniques that ought to learn the stochastic model of the dynamics and plan with DP techniques. These techniques directly inherit from the adaptive control idea. Most of the theoretical works about these algorithms study the *sample complexity* (Def. 9) of the methods [27, 69, 74, 136]. The two most known algorithms that exist are : **E3** [74] for Explicit-Exploit-or-Explore and **R-max** [27]. They both have in common the learning of the probability distribution to perform MDP planning with the learned model. Note that planning can be done in two different ways : using dynamic programming algorithms such as Value Iteration [5] or using its model to generate new samples trajectories and update in a model-free fashion, such as Dyna mode. They also share a common property : both algorithms are *PAC-MDP*, for *provably approximately correct in MDPs*, i.e. that they have convergence bounds properties. To understand what is PAC-MDP, we first need to define *Sample Complexity*.

Definition 9 (Sample Complexity [69]). Let M be an MDP with N_S states, N_A actions, discount factor $\gamma \in [0, 1)$ and a maximal reward denoted $R_{max} > 0$. Let A be a RL algorithm which acts in the environment, resulting in $s^0, a^0, r^0, s^1, a^1, r^1, \ldots$

Let $V^A_{t,M} = E[\sum_{k=0}^{\infty} \gamma^k r^{t+k} | s^0, a^0, r^0, \ldots, s^{t-1}, a^{t-1}, r^{t-1}, s^t]$ and V^* the value function of the optimal policy. Now, let $\varepsilon > 0$ be the accuracy and $\delta > 0$ be an allowed probability of failure.

The expression $\eta(\varepsilon, \delta, N_S, N_A, \gamma, R_{max})$ is a sample complexity bound for algorithm A if independently of the choice of s^0, the number of timesteps such that $V^A_{t,M} < V^* - \varepsilon$ is at most $\eta(\varepsilon, \delta, N_S, N_A, \gamma, R_{max})$ with

probability at least $1 - \delta$.

Intuitively, this quantity tells how many non-optimal (exploratory) steps does the agent make at most before convergence. We say that an algorithm with sample complexity that is polynomial in $1/\varepsilon, log(1/\delta), N_S, N_A, 1/(1-\gamma), R_{max}$ is called PAC-MDP. Finally, such algorithms differ mainly by their sample complexity due to how they treat the exploitation-exploration dilemma.

5.1.2.3 E3

The E3 algorithm scheme is as follows. The agent interacts with the environment to collect experiences for model-learning and use its model for planning. The planning process is done online, i.e. the agent performs planning immediately before executing an action so the plan drives the selected actions that will be executed in the environment. The learning process is done by counting number of visits in experiences (s, a, r, s'). The confidence about the model is given by a condition : either the agent has observed sufficiently many times the state-action pairs. This is done by maintaining a count about the visits in state-action couples $n(s, a)$. We say that a state s is known if all actions $a \in \mathcal{A}$ have been executed sufficiently many times in this state. With a given learned model $\mathcal{P}$ and $\mathcal{R}$ and the knowledge about known states, the agent builds two MDP to solve :

* *MDP known* : including all known states with the estimates $\mathcal{P}(s^{t+1}|s^t, a^t)$ and $\mathcal{R}(s^t, a^t)$;
* *MDP unknown* : including the MDP known + an absorbing aggregated state that comprise all the unknown states and where the agent receives maximum reward R_{max} (incentive for exploration in unknown states).

The process is displayed in Algorithm 24.

Algorithm 24: E3 - PAC-MDP Algorithm [74]

Input: State s
Output: Action a
Data: Learned dynamics $\mathcal{P}$ and reward $\mathcal{R}$

```
if s is a known state then
    Plan in the MDP known if return obtained from the planning above some threshold then
        Return action a from the plan                                  // Exploitation
    else
        Plan in the MDP unknown
        Return action a from the plan                                  // Planned exploration
else
    Select action a with the least observations in state s             // Pure exploration
```

5.1.2.4 Rmax

R-max [27] is a model-based reinforcement learning algorithm which can attain near-optimal average reward in polynomial time, and initially built upon the E3 scheme. In R-max, the agent always maintains a complete, but possibly inaccurate model of its environment and acts based on the optimal policy derived from this model. The model is initialised in an optimistic fashion : all actions in all states return the maximal possible reward R_{max} and the probability transition is initialised to the identity matrix such that states are in a self-loop mode at the beginning. Moreover, they define a threshold value m which serves as an indicator to consider a given state-action pair as known. On the contrary to E3, R-max solves only one unique MDP model (no separate MDP known and MDP unknown) and therefore implicitly explores or exploits. We denote by $\sim$ the learned functions of the model $\tilde{\mathcal{P}}$ and $\tilde{\mathcal{R}}$. The method is depicted in Algorithm 24.

Algorithm 25: R-max - PAC-MDP Algorithm [27]

Input: State space S, Action space $\mathcal{A}$, R_{max}, m
Output: q^*, V^*
Data: Dynamics $\mathcal{P}$ and reward $\mathcal{R}$ are unknown

Initialise $\tilde{\mathcal{R}}(s,a) = R_{max}$ and $\tilde{\mathcal{P}} = I$
Initialise all counters $n(s,a) = 0$, $n(s,a,s') = 0$ and $r(s,a) = 0$
Initialise random policy q
repeat
    Execute action $a = q(s)$ and observe s', r
    Update counters $n(s,a) \leftarrow n(s,a) + 1, n(s,a,s') \leftarrow n(s,a,s') + 1$ and $r(s,a) \leftarrow r(s,a) + r$
    /* Known state-action pair with confidence */
    if $n(s,a) \geq m$ **then**
        Update the model $\tilde{\mathcal{P}}(s'|s,a) = \dfrac{n(s,a,s')}{n(s,a)}$ and $\tilde{\mathcal{R}}(s,a) = \dfrac{r(s,a)}{n(s,a)}$ // Model-learning
        Solve the MDP($\tilde{\mathcal{P}}, \tilde{\mathcal{R}}$) to update V and q // Planning
until not converged

In addition, we can also quote the *Mormax* algorithm proposed by Szita and Szepesvári in [136] where the authors reviewed different PAC-MDP algorithms and their sample complexity bound before proposing a modified version of the Rmax algorithm.

Definition 10. In this document, we will call MDP online algorithms any model-based reinforcement learning technique where the learning agent ought to estimate the stochastic transitions and plans with dynamic programming techniques.

5.1.3 More recent model-based RL techniques

Up to now, the learned models (deterministic or stochastic) were all exact tabular functions. Although model-based RL techniques can be efficient, this modelling is not efficient in very large scale system and approximation is needed to overcome the dimension of the state or action spaces. In the precedent techniques, they only consider tabular representations of dynamics and reward models which may be impractical in large scale environments with very high size state and action spaces. More recent works [106] about model-based RL methods have been considering using approximation functions to represent the dynamics and the reward. Among them, RL methods have been proposed to approximate for example the dynamics of the model $\mathcal{P}$ by linear or non-linear approximations [108], [106]. Overall, there are multitude of approximation functions that can be implemented to represent the dynamics and the reward functions [115] :

* *Linear model* : $\mathcal{P}(s'|s,a) = \mathcal{N}(s'|w^T[s,a], \sigma^2 I)$
* *Non-linear models*
    — *Stochastic* : Gaussian processes with $\mathcal{P}(s'|s,a) = GP(s'|w^T[s,a], \sigma^2 I)$;
    — *Deterministic* : Neural networks with $s' = \mathcal{P}(s,a)$.

The scheme of a learning episode is depicted in Algorithm 26.
Finally, we leave these techniques as perspectives and won't be investigated in this document.

Algorithm 26: Model-based RL with model approximations at a given episode t [115]

Input: Q, w_P, w_R initialised
Output: q^*, Q^*
Data: True Dynamics $\mathcal{P}$ and reward $\mathcal{R}$ are unknown

Start in state $s = s^0$
repeat
  Execute action $a = \epsilon - greedy(Q, s)$ and observe s', r
  `/* Model-learning */`
  Update transition approximator weights $w_P \leftarrow w_P - \alpha_P(\mathcal{P}(s,a) - s')\nabla_{w_P}\mathcal{P}(s,a)$
  Update reward approximator weights $w_R \leftarrow w_R - \alpha_R(\mathcal{R}(s,a) - r)\nabla_{w_R}\mathcal{R}(s,a)$
  `/* Planning */`
  Update Q by planning with $\mathcal{P}(s,a), \mathcal{R}(s,a)$ `// Generate trajectories, dynamic programming`
until end of the episode
return Q,q

5.2 Multi-tier Cloud Models

The goal of this chapter is to evaluate the performances of model-based RL techniques compared to state of the art model-free Q Learning usually assessed in Cloud resource allocation scenarios [38, 68]. More precisely, we do the comparison in a multi-tier Cloud architecture where the aim is to learn the auto-scaling policy that minimise a cost function comprising energy and performance parameters.

This section presents the behaviour of a three-tier architecture model which is modeled by a tandem multi-server queuing system. We describe the associated tandem queuing system as well as the transition probabilities and the costs. Note that this multi-tier cloud model will be the use-case for the rest of the thesis, i.e. Chapter VI and Chapter VII. Moreover, it describes the same model with Markov Modulated Poisson Process (MMPP) arrivals to consider partially observable MDP (POMDP) environment and to assess the robustness of model-based RL methods.

5.2.1 Tandem Queue : Environment 1

We concentrate on a small segment of a multi-tier network with two physical nodes in tandem and consequently consider a 3-tier model (Figure 5.2). We model the 3-tier software architecture by two nodes (or multi-server stations) in tandem, where one node acts for one tier : application tier and data tier. Each node is represented by a multi server queue (or a buffer, where requests wait for a service) and servers (or Virtual Machines : VMs) which can be activated or deactivated by a controller. We assume that each node has a finite capacity, let B_1 (resp. B_2) the capacity of node 1 (respectively node 2), where the capacity of the node represents the maximum number of requests either waiting for a service or in service. Each VM is represented by a server and we define by K_1 (respectively K_2) the maximum number of usable VMs in node 1 (respectively node 2) knowing that we must have at least one machine activated. All virtual machines (VMs) in a given node are homogeneous, and the service rates can be modelled by an exponential distribution with rate μ_i for node i ($i = 1, 2$).

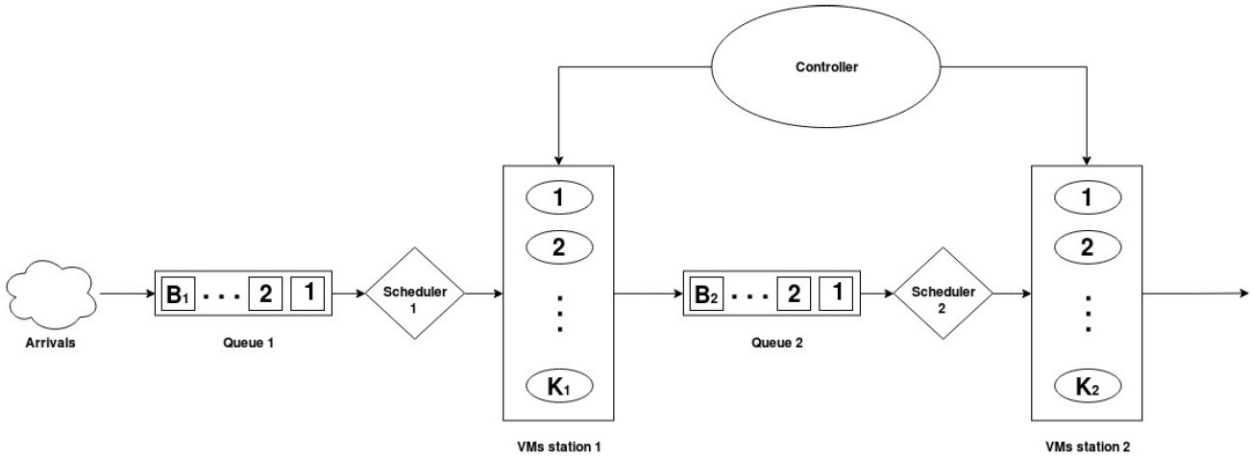


FIGURE 5.2 – Tandem queue representation of three-tier architecture

We suppose that requests arrive in the system only in node 1 and arrivals follow a Poisson process with parameter λ. When a request arrives in node 1 and finds B_1 customers in node 1, then it is lost. Otherwise, it waits in the queue until a server becomes free, then after being served in node 1 the request enters in node 2 unless the second queue is full in which case the request is lost. Once the request finishes its service in node 2, it leaves the system. At each transition epoch, a single controller manages the number of activated VM in each tier and can decide to turn on or turn off virtual machines or to do nothing. Only one VM can be deactivated or activated in each station each time. The only actions that the controller can trigger are then activation or deactivation on each node.

5.2.2 Semi Markov Decision Process Description

We detail now how the system works.

5.2.2.1 System's dynamics

The system state includes the current number of requests in node 1, denoted by m_1, in node 2, denoted by m_2, as well as the number of active servers in node 1, denoted by k_1, and in node 2, denoted by k_2. Thus, the state space is defined by $\mathcal{S}$ with $s = (m1, m2, k1, k2) \in \mathcal{S}$ such that $0 \leq m1 \leq B1, 0 \leq m2 \leq B2, 1 \leq k_1 \leq K_1$, and $1 \leq k_2 \leq K_2$.

We denote by a_i the action available in node i (with $i \in \{1, 2\}$). It can take three values : $\mathbf{1}$ if we activate one VM; $-\mathbf{1}$ if we deactivate one VM; and $\mathbf{0}$ if the number is left unchanged. Any action taken by the controller is the couple of actions in each of the nodes. Hence, the action space is $\mathcal{A}$ where $a = (a_1, a_2) \in \mathcal{A}$ with $a_i \in \{-\mathbf{1}, \mathbf{0}, \mathbf{1}\}$.

We describe now the transitions. We consider here that the controller can observe the system just after any change in the state and reacts. The actions are instantaneous. We describe now the effects of the controller's action. Its knowledge of the system as well as the way it decides which action to perform are described in Section 5.2.4. After an action, the system evolves until the next transition occurs. We define the effect of the action in node i by $N(k_i + a_i) = \min\{\max\{1, k_i + a_i\}, K_i\}$.

So, at state $s = (m1, m2, k1, k2)$, after triggering action $a = (a_1, a_2)$, the possible transitions are :
In case of an arrival in queue 1, we move, with rate λ, in :

$$s'_1 = \big(\min(m_1 + 1, B1), m_2, N(k_1 + a_1), N(k_2 + a_2)\big) .$$

In case of departure from queue 1 and entry in queue 2, we move, with rate $\mu_1 \min(m_1, N(k_1 + a_1))$ in :

$$s_2' = \big(\max(m_1 - 1, 0), \min(m_2 + 1, B_2), N(k_1 + a_1), N(k_2 + a_2)\big).$$

In case of departure from queue 2, we move, with rate $\mu_2 \min(m_2, N(k_2 + a_2))$, in :

$$s_3' = (m_1, \max(m_2 - 1, 0), N(k_1 + a_1), N(k_2 + a_2)) .$$

We define the transition rate of state-action pair (s, a) by $\Lambda(s, a)$ such that :

$$\Lambda(s, a) = \lambda + \mu_1 \min\{m_1, N(k_1 + a_1)\} + \mu_2 \min\{m_2, N(k_2 + a_2)\} .$$

We finally have the following transition probability structure derived from the infinitesimal generator $\mathcal{Q}$. We describes the embedded markov chain associated to the transition structure as :

$$p\left(s' \mid s, a\right) = \begin{cases} \frac{\lambda}{\Lambda(s,a)} \text{ if an arrival occurs : } s' = s_1' \\ \frac{\min\{m_1, N(k_1+a_1)\}.\mu_1}{\Lambda(s,a)} \text{ if a departure occurs : } s' = s_2' \\ \frac{\min\{m_2, N(k_2+a_2)\}.\mu_2}{\Lambda(s,a)} \text{ if a departure occurs : } s' = s_3' \\ 0 \text{ otherwise with : } s' = s \end{cases}$$

5.2.2.2 System's costs

We consider a continuous time discounted model [5] with the discount factor γ. We define the costs that represent the trade-off between quality of services (*QoS*) and energy consumption. There are *instantaneous costs* that are charged only once where C_A denotes the activation cost of a VM, C_D its deactivation cost and C_R the cost of rejecting a request. There are *accumulated costs* that accumulate over time : C_S denote the cost per time unit of using a VM and C_H the cost per time unit of holding a request in the system. In [141] is presented a way to give values to these costs so that they have a real meaning regarding cloud infrastructure. Formally, after triggering action $a = (a_1, a_2)$ in state $s = (m_1, m_2, k_1, k_2)$, the accumulated cost c_i at node i equals :

$$c_i(s, a) = N_i(k_i + a_i) \cdot C_S + m_i \cdot C_H$$

and the instantaneous cost at node i is equal to :

$$h_i(s, a) = C_A \cdot \mathbb{1}_{\{a_i=1\}} + C_D \cdot \mathbb{1}_{\{a_i=-1\}} + \frac{\lambda}{\Lambda(s, a) + \gamma} C_R \mathbb{1}_{\{m_i=B_i\}} \text{ if } i = 1;$$

$$h_i(s, a) = C_A \cdot \mathbb{1}_{\{a_i=1\}} + C_D \cdot \mathbb{1}_{\{a_i=-1\}} + \frac{\min\{m_1, N(k_1 + a_1)\} \cdot \mu_1}{\Lambda(s, a) + \gamma} C_R \mathbb{1}_{\{m_i=B_i\}}$$

if $i = 2$.

Terms in front of the reject cost C_R comes from the probability that the event is an arrival and the buffer is full.

We consider here a continuous time discounted model. The discount factor is denoted by γ. We define a Markov Deterministic stationary policy q as a mapping from state space to action space $q : \mathcal{S} \to \mathcal{A}$. This mapping defines the action to be performed in a given state s. From discounted model in [5], we define the stage cost function $\mathcal{R}$ as :

$$\mathcal{R}(s, a) = \frac{1}{\Lambda(s, a) + \gamma} \left[c_1(s, a) + c_2(s, a)\right] + h_1(s, a) + h_2(s, a).$$

5.2.2.3 Uniformisation

Most reinforcement learning models and more precisely model-based applications deal with discrete time scenario. In order to handle this control model with an RL approach we will uniformise the continuous time model to obtain a discrete time model on which we can use standard methods of R.L. some of which are already implemented in R.L. libraries.

The way to proceed (see details in chapter 11 of [5]) is first to define a constant which is finite and larger or equal to the maximum transition rate. Here we take $\tilde{\Lambda} = \lambda + K_1 \cdot \mu_1 + K_2 \cdot \mu_2$. Then, for each state-action couple (s, a) we add a transition associated with a pseudo event so that the process remains in the same state s. In (s, a), this transition has rate $\tilde{\Lambda} - (\Lambda(s, a))$ thus the transition rate is constant with rate $\tilde{\Lambda}$ regardless (s, a). The transition probabilities in the uniformised model are denoted by $\tilde{p}(s'|(s, a))$ and are given by :

$$\tilde{\Lambda} \times \tilde{p}(s'|s, a) = \begin{cases} \lambda & \text{if } s' = s'_1 \\ \mu_1 \min\{m_1, N(k_1 + a_1)\} & \text{if } s' = s'_2 \\ \mu_2 \min\{m_2, N(k_2 + a_2)\} & \text{if } s' = s'_3 \\ \big(\tilde{\Lambda} - \Lambda(s, a)\big) & \text{when } s' = s \\ 0 & \text{otherwise .} \end{cases}$$

The states s'_2 and s'_3 are defined in the system's dynamic part above. It exists only few states for which we naturally jump into the same state (*i.e.* $s' = s$). These states are such that $m_1 = B_1$ and $N(k_1+a_1) = k_1$ and $N(k_2+a_2) = k_2$ and the event is an arrival. In such cases the transition probability toward $s' = s$ is equal to $\tilde{\Lambda} - (\Lambda(s, a) - \lambda\big)/\tilde{\Lambda}$, see [5].

The stage costs also need to be modified. Now they are given in the discounted model by :

$$\tilde{\mathcal{R}}(s, a) = \frac{\Lambda(s, a) + \gamma}{\tilde{\Lambda} + \gamma} \mathcal{R}(s, a) .$$

Then the Bellman Equation of such a model is :

$$V^*(s) = \min_{a \in \mathcal{A}} \big(\tilde{\mathcal{R}}(s, a) + \frac{\tilde{\Lambda}}{\tilde{\Lambda} + \gamma} \sum_{s'} \tilde{p}(s'|(s, a)) V^*(s') \big) .$$

Note that this equation is never solved in R.L. methods but we solve it in our numerical experiments to assess the precision of the algorithms. Indeed its solution is the theoretical optimal value.

5.2.3 MMPP Tandem Queue model : Environment 2

In this section we want to assess robustness of reinforcement learning algorithms and especially model-based algorithms. For this purpose, we consider variations in the requests arrival rate with Markov Modulated Poisson Process system. However, these variations would be remained ignored by the agent which expects a constant intensity. We study how the algorithms react to sudden and large increases of packet arrivals and if they can quickly adapt their policy to overcome bursts traffic. This is a key element in network in which statistics are often not very precise. This is all the more important for model-based methods, since it was quoted in [46] that offline policies might no be adequate as soon as there exist changes in the dynamics of the environment. It is believed that this lack of flexibility comes from the offline learning of the policy in model-based methods and that this offline learning is done from a predefined model which does not allow to adapt to changes in the dynamics of the environment.

5.2.3.1 Markov Modulated Poisson Process

We modify now the arrival process. We want to integrate a variability feature and we consider a Markov modulated Poisson process (MMPP) where arrival rates vary over time. The process switches between different Poisson process which differ by their intensity indicated by their arrival rate λ_j. We assume we have J phases, each of them corresponding to a specific arrival rate λ_j. The switch between the phases follows a continuous time Markov chain. It is represented by a birth and death process [125] with rates $\mathcal{Q}$. We move from phase j to phase $j+1$ with rate $q_{j,j+1}$ and from phase j to phase $j-1$ with rate $q_{j,j-1}$. Usually [125], $q_{j,j+1}$ and $q_{j,j-1}$ for all j are much smaller than arrival rates $\{\lambda_j\}_j$. We extend the previous state $s = (m_1, m_2, k_1, k_2)$ to a new state representation integrating the phase in the SMDP :

$$s = (j, m_1, m_2, k_1, k_2)$$

with λ_j the current arrival rate in phase j.

In our model, we assume that $J = 2$. The first phase is considered as *normal* and the second phase called *burst* is a phase with a very high intensity. We denote by $q_{1,2}$ the transition rate from phase 1 to phase 2 and by $q_{2,1}$ the transition rate from phase 2 to phase 1.

MMPP System's dynamics We need to integrate new phase transitions in the whole environment dynamics. Under given phases j, we have the dynamics detailed in section 5.2.2.1 for arrival rate λ_j. The system can have two additional events corresponding to a change of its phase, *i.e.* the current arrival rate changes. Thus :
We move :

from $s = \big(j, m_1, m_2, N(k_1+a_1), N(k_2+a_2)\big)$ with rate $q_{j,j+1}$ to $s'_4 = (j+1, m_1, m_2, N(k_1+a_1), N(k_2+a_2))$;

from $s = \big(j, m_1, m_2, N(k_1+a_1), N(k_2+a_2)\big)$ with rate $q_{j,j-1}$ to $s'_5 = (j-1, m_1, m_2, N(k_1+a_1), N(k_2+a_2))$.

We define the new transition rate by state by $\Lambda(s,a)$ such that :

$\Lambda(s,a) = q_{j,j+1} + q_{j,j-1} + \lambda_j + \mu_1 \min\{m_1, N(k_1+a_1)\} + \mu_2 \min\{m_2, N(k_2+a_2)\}$. The cost function remains the same since variation in the arrival rate does not induce additional costs.

MMPP Uniformisation For the MMPP uniformisation process we take $\tilde{\Lambda} = \max_j q_{j,j+1} + \max_j q_{j,j-1} + \max_j \lambda_j + K_1 \cdot \mu_1 + K_2 \cdot \mu_2$. Then, for each state-action couple (s,a) we add a transition associated with a pseudo event so that the process remains in the same state s. In (s,a), this transition has rate $\tilde{\Lambda} - \Lambda(s,a)$ such that the transition rate of the whole point process is constant with rate $\tilde{\Lambda}$ regardless (s,a). The transition probabilities in the uniformised model are denoted by $\tilde{p}(s'|(s,a))$ and satisfy :

$$\tilde{\Lambda} \times \tilde{p}(s'|s,a) = \begin{cases} \lambda_j & \text{if } s' = s'_1 \\ \mu_1 \cdot \min\{m_1, N(k_1+a_1)\} & \text{if } s' = s'_2 \\ \mu_2 \cdot \min\{m_2, N(k_2+a_2)\} & \text{if } s' = s'_3 \\ q_{j,j+1} & \text{if } s' = s'_4 \\ q_{j,j-1} & \text{if } s' = s'_5 \\ \big(\tilde{\Lambda} - \Lambda(s,a)\big) & \text{when } s' = s \\ 0 & \text{otherwise} \end{cases} .$$

Similarly as before, the states for which it exists a natural jump into the same state are those already described for which the event is an arrival. In such a case the transition probability toward $s' = s$ is equal to
$\Big(\tilde{\Lambda} - (\Lambda(s,a) - \lambda_j)\Big) / \tilde{\Lambda}$.

5.2.4 Simulated SMDP environment and objective function for RL agent

Now, we first provide how the agent perceives and interacts with the simulated environment, next display the objective function that the learning agent ought to optimise.

5.2.4.1 Simulated SMDP environment and agent's observation

We are dealing with a reinforcement learning model therefore the uniformised model of Section 5.2 is not known but only experienced by the controller. Indeed, the controller does not have any information about queuing statistics (arrival rate, etc.), therefore does not know the dynamics of the system. The environment has state space $\mathcal{S}$ and action space $\mathcal{A}$ similarly to the SMDP model. When the agent interacts with the environment, the SMDP model is simulated. It will behave by returning a state, sampled according to the transition probabilities $\mathcal{P}$, and return the costs $\mathcal{R}(s, a)$. The cost and the new state are the only information that the controller will discover.

For the partially observable scenario described with the MMPP environment, the agent's observation remains the same. As already said, we consider a model with variation of arrivals, thus the system behaves as the SMDP described just above and the system state is described by $s = (j, m_1, m_2, k_1, k_2)$. However, we assume that the learning agent ignores these variations. Hence, the system can be only observed partially and the environment state is described by $o = (m_1, m_2, k_1, k_2)$. In this way, the two system states $(1, m_1, m_2, k_1, k_2)$ and $(2, m_1, m_2, k_1, k_2)$ translate in the same environment state (m_1, m_2, k_1, k_2). The goal is to assess and compare the robustness of reinforcement learning algorithms in the context where the agent does not have knowledge of an explanatory variable.

Overall, we consider in this work countable discrete state space where the agent can evaluate value function, policy, transition and reward matrices with tabular forms. We want to minimise the expected discounted cumulative costs.

5.2.4.2 Objective function

We search the best policy q to minimise the expected discounted expected reward, therefore the objective function is :

$$V^*(s) = \min_q \mathrm{E}^q \left[\sum_{k=0}^{\infty} \exp^{-\gamma t_k} \mathcal{R}(s_k, q(s_k)) \mid s_0 = s \right], \tag{5.1}$$

with t_k the epoch and s_k the state of the kth transition.

5.3 Generalisation and Selection of Model-based Reinforcement Learning Algorithms

In this chapter, we want to compare several tabular model-based RL algorithms with state of the art model-free algorithm, such as tabular Q-Learning, in the tandem queue scenario. We want to assess if the learning agent can benefit from learning the model underlying the environment and if it is worth increasing the complexity to speed up learning convergence. After selecting the model-based RL techniques, we present an aggregated pseudo-code for model-based reinforcement learning and discuss the parameterisation of the methods, with respect to the items presented in Chapter III (breadth planning, depth planning, integration learning and planning, etc.). Moreover, we assume here that the reward function is provided to the agent, thus it only needs to update the dynamic model $\mathcal{P}$.

5.3.1 Algorithms selection

For comparison of model-based Reinforcement Learning with SoTA model-free techniques we have selected the following methods :

* *Deterministic models - buffer replay* : Dyna architectures with *Dyna-Q*, *Dyna-Q*-+ and Dyna-Q with prioritisation on the TD-error in the buffer replay, denoted *Dyna-Q-prior*;
* *Stochastic model* : *MDP online*, an algorithm that learns the stochastic transitions to perform DP algorithms such as VI or PI, inspired by Rmax and E3 techniques.

In this part, we describe the different elements on which we can operate in the model-based algorithms. Indeed, many factors affect the quality of learning : the frequency of model update, the frequency of planning, and the planning itself (breadth, depth, etc.).

5.3.2 Learning process

The learning process is different in deterministic and stochastic models. For Dyna architectures, the learning process consists of storing the experiments (s, a, r, s') into the replay buffer, and for the prioritised version to add the TD-error for each tuple $\delta(s, a)$ in order to sort the samples for planning. On the other hand, the learning in the stochastic process is made by counting occurrences of samples and updating the transition probability for state-action pairs as explained in Section 5.1 [23] :

$$\mathcal{P}(s'|s, a) = \frac{\sharp(s, a, s')}{\sharp(s, a)}$$

5.3.3 Planning process

We encourage the readers to take over Figure 3.3 to see the difference between different approaches. In the Dyna architectures during the planning phase, the learning agent reuse its past experiences to simulate trajectories for supplementary update of the Q-value function. As we have seen in Section 5.1, the planning process has several parameters, regarding the number of trajectories it generates, the breadth and depth of the process. In our implementation, the agent will generate several rollouts starting from a state s encountered in the buffer and will proceed a single time step (depth ξ equal to 1) by retrieving the next state s' and reward r from the memory. The number of rollouts (or breadth) is denoted ζ. In *Dyna-Q* and *Dyna-Q*-+, the sampling of state s is done randomly while it is carefully done in *Dyna-Q-prior*. In this last deterministic model algorithm, the same process is done only the selection of states s is made by taking the samples with the highest TD-error. In the implementation process, this is done by sorting decrasingly the tuples $(s, a, r, s', \delta(s, a))$.

On the other side, the *MDP online* method plans using DP algorithms such as Value Iteration on the learned model $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$.

5.3.4 Integration planning and learning

To integrate the learning and the planning phases, we provide two parameters that basically gives the frequency on which the agent should update its model or plan. We refer to ν as the frequency on which the agent should update its model. Note that this is only for the stochastic models approach since model-learning in Dyna architectures only consist in storing the experiment in the memory. Knowing when to update the model is a complex task. Indeed, updating the model too often will increase complexity and, during the early phase of learning, agent will plan using a very poor precision approximated model. Moreover, we denote β the frequency on which the agent should plan with its learned model and update its value function or policy. Finally, for the exploitation

versus exploration dilemma, we have chosen the ϵ-greedy policy to drive the acting process in the environment, that is used in Dyna architectures presented by [8]. This technique will be applied to all assessed model-based techniques.

5.3.5 Parameters of model-based RL techniques

The generalised model-based algorithm is described in Algorithm 27. Overall, all model-based methods works similarly : they simulate several episodes for which the agent performs several iterations (s, a, r, s'). The model-learning phase depends on the type of the algorithm (either it stores the samples in the buffer as a deterministic model, either it updates the stochastic transition matrix with the collected data). At times, it performs planning by simulating trajectories with the learned models : with DP techniques for MDP online and rollouts with specific breadth and depth for Dyna techniques. At the end of each episode, we decrease the value of ϵ such that the agent is increasingly tempted to exploit its policy. The comparison criteria will be further discuss in Section 5.4.

We provide a reminder of notations for the model-based RL algorithms parameters that occur in all techniques :

* Integration Learning and Planning :
  - *Model learning frequency* : ν
  - *Planning frequency* : β
* Planning complexity with simulated trajectories :
  - *Breadth* : ζ
  - *Depth* : ξ

We also mention that having the learned transition probabilities $\mathcal{P}$ could serve as a predictor to simulate trajectories such as in Dyna algorithms yet we will only run DP algorithms with MDP online technique in this chapter. We resume in Table 5.1 the RL methods that will be evaluated on the two tandem queue systems. Notice that different parameterisation of the algorithms such as breadth and depth planning, model frequency update and planning frequency will be assessed in the experimental results.

Algorithms	Model	Type of Model	Planning	Bonus exploration
Q-Learning	/	/	/	/
Dyna-Q	Experience samples	Deterministic	Trajectories with random experiments	/
Dyna-Q-+	Experience samples	Deterministic	Trajectories with random experiments	κ
Dyna-Q-prior	Experience samples	Deterministic	Trajectories with prior experiments	/
MDP online	Tabular $\mathcal{P}, \mathcal{R}$	Stochastic	Dynamic Programming	/

TABLE 5.1 – Summary of selected model-based RL algorithms

5.4 Experimental Results

This section presents the experimental results of RL algorithms comparison in the simulated tandem queue systems. We first discuss the comparison criteria that we have selected to efficiently and fairly compare the performances of the algorithms. Next, we describe the tandem queue simulator and environment parameters. Finally, we display the results for the initial tandem queue system, followed by comparison in the MMPP scenario.

Algorithm 27: Generalised Model-based reinforcement learning

Input: V_0, Q_0, ν : frequency model update, β : frequency planning, ξ : planning depth, ζ : planning breadth, ϵ : exploitation vs exploration

Output: q^*, Q^*

Data: System dynamics $\mathcal{P}$ unknown

```
/* Loop until end of episodes                                              */
```

for $e \in MaxEpisode$ **do**

  Select state $s_0 \in \mathcal{S}$ // Initial state

  for $i \in MaxIteration$ **do**

    Take action $a \in \mathcal{A}$ with ϵ-greedy policy // Action selection

    Observe s' and reward $r(s,a)$ and store tuple (s,a,r,s') // Deterministic model-learning

    Model-free update $Q(s,a) = Q(s,a) + \alpha\left[r(s,a) + \gamma \max'_a Q(s',a') - Q(s,a)\right]$

    /* Model-learning frequency */

    if $i \,\%\, \nu = 0$ **then**

      Update transitions probabilities $\mathcal{P}(s'|s,a)$ // Update model $\tilde{M}$

    /* Planning frequency */

    if $i \,\%\, \beta = 0$ **then**

      /* For Dyna architectures */

      for $b \in 1, .., \zeta$ **do**

        Select (s,a) from replay buffer

        for $p \in 1, ..., \xi$ **do**

          $s', r \leftarrow$ Buffer(s,a) // With buffer-oriented methods

          Q learning update $Q(s,a)$

      /* For MDP online */

      Solve the MDP with the learned model $\mathcal{M}$: VI or PI

5.4.1 Comparison Criteria between RL Algorithms

Directly adapting methods to compare algorithms used by practitioners is proving to have little relevance and utility. We adopt the guidelines proposed in [34] as a guide for rigorous comparisons of reinforcement learning algorithms. Comparisons are divided into two main parts : comparison while learning and comparison after learning on test environment.

5.4.1.1 Learning curves comparison

First, we compare different algorithms during the learning phase. Instead of only comparing the final performances of the RL methods after t timesteps in test environment, we can compare performances along learning. Indeed, performance measures while learning are represented by a learning curve. This reveal differences in speed of convergence and can provide more robust comparisons. In more detail, the learning curve will display the average discounted reward obtained by the algorithms at the end of each learning episode. This will give the evolution of the average reward obtained and will show how fast the algorithm learns a good policy, therefore displaying the speed of convergence. Since our problem is a cost minimisation problem, best performances will come from algorithm that can quickly decrease the average discounted reward.

5.4.1.2 Test policy comparison

RL algorithms should also be assessed offline. The algorithm performance after t iterations is measured as the average of the returns over N evaluation episodes conducted independently after training, similarly as a Monte Carlo policy evaluation. The evaluation is done implementing the policy q returned by the RL method at step t. Again, we look at the average discounted reward obtained by the plugged-in policy for comparison between different algorithms.

5.4.1.3 Comparison criteria used in our experiments

Based on literature and experimental considerations, we devised our own comparison criteria, mainly for the learning comparison between algorithms. We express two comparisons regarding learning curves. The first one is the average reward obtained after each learning episodes e. For this purpose, we store at the end of each episode the average discounted reward obtained by the algorithm and plot at the end of the learning process the learning curve that depicts the average reward over all episodes. This will show how fast the algorithms can optimise the average reward obtained by the updated policy, i.e. how fast the convergence is and how quickly RL algorithms can adapt the policy in environment with changes such as Section 5.2.3.1.

After learning, we evaluate the learned policy of each algorithm in a test environment. We evaluate all policies in Monte Carlo simulations (starting from a state s_0) for 50 episodes of 10000 iterations. We finally take the mean discounted reward obtained overall and can compare the goodness of the algorithms.

5.4.2 Simulation Environment and Parameters

To compare the list of RL algorithms, described in Section 5.3 Table 5.1, we implemented a python software to simulate the two tandem queue systems, that serves as the environment in which the agent interacts with and collect data.

5.4.2.1 Gym environment

OpenAI Gym [29] is a toolkit for developing and comparing reinforcement learning algorithms. We developed a python simulator under a Gym environment. Two environments were implemented : **Environment 1** represents the tandem queue model and **Environment 2** represents the model with MMPP. The implementation of the environment class is made with two principal methods :

— *Reset* : Reinitialise the environment with a new state $s = (m_1, m_2, k_1, k_2)$;

— *Step(action)* : Return the new state s' and the reward r after the agent has done action a in state s by simulating the uniformised probability and reward of the tandem queue model.

5.4.2.2 Cloud parameters

We ran multiple simulations on both environments. We first describe cloud simulations parameters that are common to all experiments. We considered three main cloud scenarios :

* *C1* : $B_1 = B_2 = 5, K_1 = K_2 = 3$, $\lambda = 8$, $\mu_1 = 2, \mu_2 = 2$;
* *C2* : $B_1 = B_2 = 20, K_1 = K_2 = 5$, $\lambda = 15$, $\mu_1 = 2, \mu_2 = 2$;
* *C3* : $B_1 = B_2 = 30, K_1 = K_2 = 8$, $\lambda = 15$, $\mu_1 = 2, \mu_2 = 2$.

The cost parameters remain the same for all Cloud scenarios : $\{C_a = 5, C_d = 5, C_s = 10, C_h = 10, C_r = 100\}$.

5.4.2.3 Learning parameters

The learning phase runs for 50 episodes of length 10000 iterations. One iteration corresponds to a transition from state s with action a to state s' with reward r. We collect at the end of each episode the average discounted reward obtained by the learning agent while interacting with the environment. The agent runs an epsilon-greedy policy over the whole learning process, with initial epsilon set to $\varepsilon = 1$ and epsilon decay $\varepsilon_{dec} = 0.95$. We decrease epsilon value after each episode. The discount rate is set to $\gamma = 0.9$.

5.4.2.4 Algorithms parameters

For the first series of simulation, we provide a single set of parameters for the model-based RL algorithms. This includes frequency update of the model (for *model* algorithms, frequency of planning phases and the depth of the planning phase (number of planning iterations).

Parameters for Dyna architectures For the Dyna architectures, the agent does the planning after each iteration, i.e. $\beta = 1$ and perform a one-step sampling for 10 samples stored in the buffer, i.e. $\zeta = 10$ and $\xi = 1$. The model learning is also done after each iteration by storing the samples (s, a, r, s') in the buffer memory, i.e. $\nu = 1$.

Parameters for MDP online The parameters are different in the MDP online algorithm. The model learning is done at the end of each episode. The agent updates the whole transition matrix $\mathcal{P}(s'|s, a)$ for all couples (s, a, s') with the collected data. The model-learning frequency is thus $\nu = MaxIteration$. For the planning part, it is also done at the end of each episode. Once the agent have updated its MDP model with the dynamics $\mathcal{P}$, it solves the MDP with the Value Iteration algorithm to update its value function V and the policy q. The breadth and depth are therefore the ones of DP algorithms as displayed in Figure 3.3.

5.4.3 Experimental Results for Initial Tandem Queue Environment

First, we display experimental results for the Environment 1, i.e. the initial tandem queue scenario. We first look at the learning curves to compare the speed of convergence of algorithms in the Cloud scenarios *C1, C2, C3*. Before showing the results, we first provide some preliminary results in Figure 5.3 showing that the Dyna-Q-+ algorithm was not providing improvements in this situation. Indeed, we can observe that it performs similarly as the original Dyna-Q algorithm. For this purpose, the rest of the experimental results are conducted only with Q-Learning, Dyna-Q, Dyna-Q-prior and MDP online.

FIGURE 5.3 – Average reward over learning episodes in tandem queue environment with larger scale

5.4.3.1 Learning Curves Comparison

We conducted the experimental results on the three Cloud scenarios. Figures 5.4, fig :asmtaRes2, fig :asmtaRes3 respectively show the average discounted reward obtained over the learning process by the different algorithms on the three Cloud scenarios *C1, C2, C3*. Moreover, we display a black threshold line corresponding to the optimal policy computed from an oracle agent that solves the MDP with DP techniques. We observe in these Figures a small gain from Dyna architectures compared to Q-Learning. Dyna-Q accelerates very slightly the convergence (red line below the blue line) but this can be improved by increasing the breadth and depth for the planning. We also notice that the Dyna-Q-prior performs better than then original Dyna-Q, demonstrating the gain of prioritising the samples used in the planning process. Last, what strikes us when we see these curves, are the performances of the MDP online algorithm which clearly outperforms the others and can even reach the optimal policy.

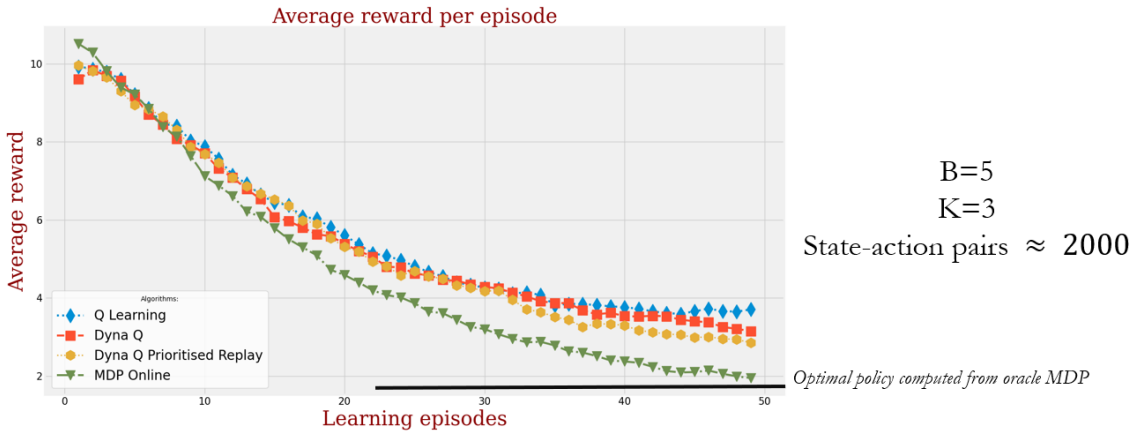


FIGURE 5.4 – *C1* average discounted reward obtained by RL algorithms

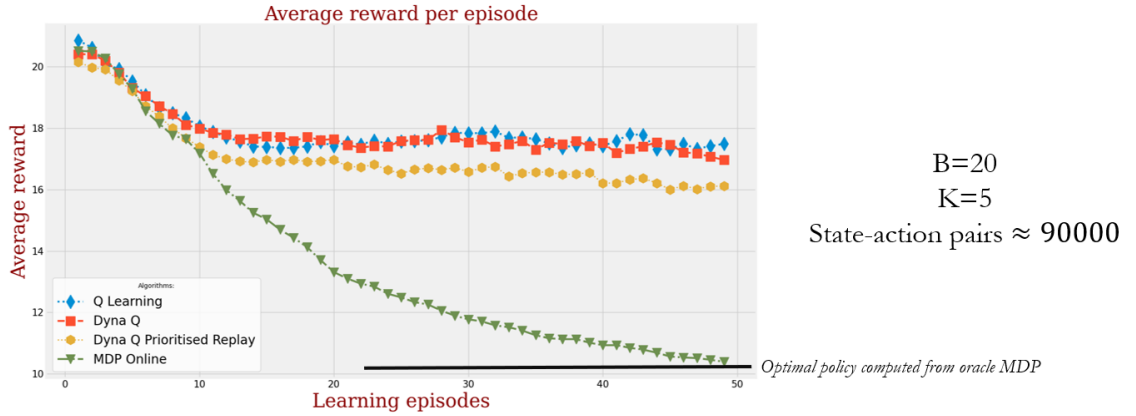


FIGURE 5.5 – *C2* average discounted reward obtained by RL algorithms

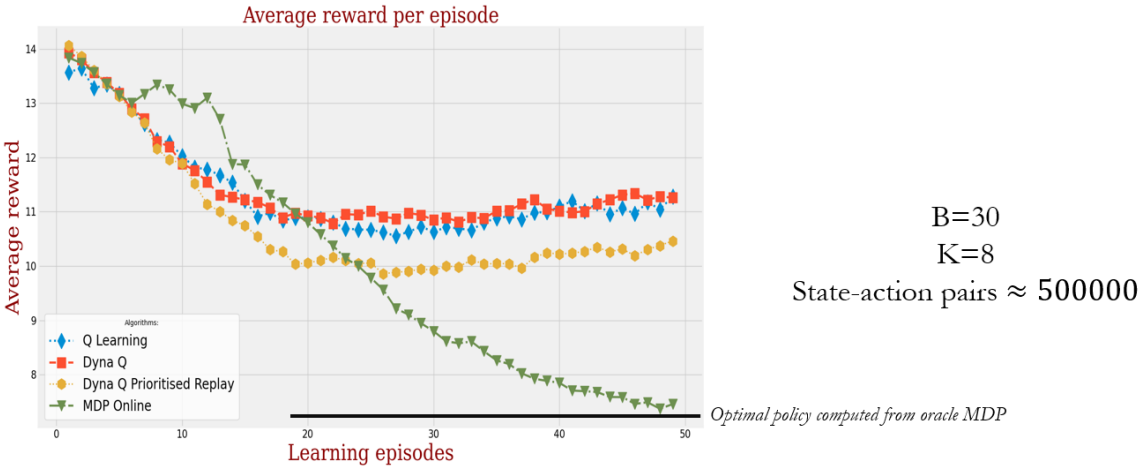


FIGURE 5.6 – *C3* average discounted reward obtained by RL algorithms

Now, these results need to be completed by more simulations. First, by acting on the different algorithms parameters for planning and model-learning. Secondly, these simulations are conducted for a same amount of

iterations in the system. However, we seek to consider two quantitative elements : the quantity of interactions as well as the execution time. Indeed, these are two important elements for a machine learning analyst. It must be noticed that, due to the offline updates, the complexities of the model-based algorithms are greater than these of model-free algorithms. Thus model-based algorithms have a larger running time for the same number of interactions with the environment. Henceforth, we seek to provide two comparisons : one where the number of interactions are the same between model-based algorithms and Q-learning and a second one where the running times are the same.

5.4.4 Experimental Results for MMPP Tandem Queue

Now, we display preliminary experimental results for the second Cloud environment : the MMPP scenario.

5.4.4.1 Environment characteristics

We consider a scenario where burst arrivals can appear. The goal is to assess robustness of RL algorithms to sudden changes in the system. Arrival rates λ_j can take two values : $5, 30$. We define transition rate between phases by : $q_{0,1} = 1$ and $q_{1,0} = 10$. This leads to low chances to have burst phase and when it happens to remain in this phase a very short amount of time. We did a first comparison on a small Cloud scenario where $B_1 = B_2 = 10, K_1 = K_2 = 3$ and service rates were $\mu_1 = \mu_2 = 2$.

5.4.4.2 Policy evaluation

We first show a comparison between Q-Learning with the Dyna architectures and demonstrate that these model-based techniques with deterministic models suffer from variation in queuing statistics. We can see in Table 5.2 that the average discounted reward measured after a Monte Carlo simulation is better in model-free technique compared to model-based ones for a same learning time. One explanation is that model-based RL agent keeps updating its value function based on an outdated model of the world. Therefore, they do not integrate new dynamics change and blend model-free update with model-based update resulting in poor performances. These findings confirm theoretical assumptions about these model-based techniques for varying environment. However, we can notice that in this environment, the Dyna-Q-+ performs better than the others.

Algorithms policy	Average discounted reward
Q-learning	2,63
Dyna-Q-buffer	3,05
Dyna-Q-buffer-plus	2,92
Dyna-Q-buffer-prior	2,98

TABLE 5.2 – Average discounted reward obtained by a Monte Carlo policy evaluation after a fix period of learning

5.4.5 Final comparison in two multi-tier environments

To conclude on the reinforcement learning algorithms comparison in both environments, we provide a summary table in 5.7 with comparisons on both initial tandem queue environment and MMPP environment. We display the mean distance between average reward obtained in the learning episodes and the average reward calculated from the optimal MDP policy. This distance is displayed at in the tables for different episodes (25,50 and 100). We demonstrate that in the first initial tandem queue environment, the MDP online algorithm has the best performance regarding speed of convergence and confirm the sample efficiency of such approaches. Moreover, the

Dyna approaches perform slightly better than the model-free Q-Learning , however take more time to converge to the optimal solution.

On the contrary, for the MMPP tandem queue environment, the results are different. First, we observe a deterioration of the performances in all RL algorithms due to the partially observable setting of this environment. The MDP online algorithm remains the best technique in this scenario as it allows the RL agent to learn the stochastic transition matrix and have an idea about the mean value of different arrival rates. Last, we observe a huge deterioration of Dyna architectures that obtain less performances than the Q-Learning. One possible explanation is that the learning of a deterministic model could lead to poor performance since the agent will plan with an old model of previous environment transitions while the environment has changed (burst phase, etc.).

Initial tandem queue:

Mean distance between average reward in the learning episode with average reward of the optimal MDP policy

~ 50 Cloud instances	Q LEARNING	DYNA Q	DYNA Q PRIOR	MDP ONLINE
Episode 25	15,37	15,12	14,57	12,78
Episode 50	11,33	10,68	9,37	6,2
Episode 100	**5,23**	4,11	3,7	**0,3**

MMPP tandem queue: (with bursty arrivals every 100 timesteps)

~ 30 Cloud instances	Q LEARNING	DYNA Q	DYNA Q PRIOR	MDP ONLINE
Episode 25	16,38	16,47	16,18	14,68
Episode 50	12,1	12,51	12,43	8,37
Episode 100	7,12	**8,35**	7,9	**3,15**

FIGURE 5.7 – Comparison of RL algorithms in two tandem queue environments

5.5 Summary of the chapter

In this chapter, we first have shown that on the initial tandem queue environment, model-based RL methods could provide a significant gain regarding the convergence speed. Although Dyna architectures performs slightly better than the Q-Learning, the MDP online algorithm outperforms all the assessed techniques. This demonstrates the sample-efficiency and faster convergence of deterministic and stochastic model-based RL algorithms. Moreover, more simulations are conducted for a fair evaluation of the algorithms on different learning parameters and algorithms parameters. On the partially observable system with varying arrival rates however, the deterministic models are suffering as expected, where on the other hand stochastic models can handle efficiently the changes in the system's statistics. Last, we underline that the approach presented here can be applied to more general network models and with a large set of distributions for arrivals or services and even directly by using traffic traces of cloud platforms. We expect these results would be still valid in these cases.

Furthermore, we did not study the policies structure in this chapter as we did in Chapter IV. It is therefore a track to consider since it has been studied in the literature the structure of value functions and policies in tandem queue system [94]. This structural properties can be integrated in the RL algorithms to accelerate the convergence.

Now, the major issue in this chapter is the consideration of relatively small Cloud environments due to difficult representation and implementation of tabular approaches in large scale systems. Although one direction is to

consider approximation approaches such as described in Section 5.1, we decided next to investigate structural approaches and to assess if providing more knowledge about the environment to the agent could help to represent more compactly the world model and also could help for convergence acceleration. These approaches are explored in the following chapters with Factored RL Chapter VI and Causal RL Chapter VII.

PART III

MODEL-BASED REINFORCEMENT LEARNING WITH RELATIONAL STRUCTURE BETWEEN ENVIRONMENT VARIABLES

CHAPTER 6

FACTORED REINFORCEMENT LEARNING FOR MULTI-TIER NETWORK

The contributions of this chapter have been published in [140] :

* Thomas Tournaire, Yue Jin, Armen Aghasaryan, Hind Castel-Taleb, Emmanuel Hyon. Factored Reinforcement Learning for Auto-scaling in Tandem Queues. In *Network and Service Management in the Era of Cloudification, Softwarization and Artificial Intelligence (NOMS 22)*, 2022

In the precedent chapters, we only considered tabular representations of dynamics and reward models which may be impractical in large scale environments with very high size state and action spaces. As we have seen at the beginning of this work, the *factored MDP* framework ought to represent more compactly the model $\mathcal{M}$ of the world by integrating the relations between environment variables. In environments which have specific relations between state variables, it could be highly beneficial to use this representation to accelerate the convergence and the ease of implementation.

This chapters proposes **FMDP online**, a factored RL algorithm that can overcome convergence issues by expressing the problem in a compact form. Indeed, all the solutions described in the MDP framework of chap IV and chap V, whether for planning or reinforcement learning with models, all share a common drawback : they are not adapted to the resolution of large problems. The use of unstructured representations such as tables for dynamics and value function requires an explicit enumeration of the set of all the possible states of the problem to represent the functions necessary to solve it.

The factored approach provides the learning agent with the relational structure of the environment, here with Dynamic Bayesian Networks (DBN). Indeed, we believe that in distributed network environments such as $5G$ slicing architectures, queuing networks or multi-tier infrastructures presented here with the tandem queue model, there exist local dependencies between physical nodes. Events in a physical node have direct impacts only on its neighbours and affect other nodes indirectly in most cases. In this context *Factored MDP* [26] framework takes advantage of local dependencies structure in the environment, accelerates convergence and overcomes the so-called 'curse of dimensionality' in MDP solutions. We use the Factored MDP framework to model the auto-scaling problem in the tandem queue with two nodes, described in section 5.2.1. The very few number of applications with factored solutions in the literature call for further research in this topic and for comparison with existing model-free and model-based algorithms. To the best of our knowledge, we have not seen works that apply factored RL approaches on networking systems and this is why we wanted to investigate such approach in a feasible environment.

Contributions of this chapter are as follows :

* Proposal of FMDP online algorithm;
* Factored MDP model for the tandem queue environment;
* Comparison of FMDP online with state of the art algorithms on the tandem queue model.

6.1 Background

We first provide more details about the Factored Markov Decision Process (FMDP) framework that was briefly introduced in Chapter III, by providing mathematical formalism and state of the art algorithms.

6.1.1 Coffee Robot Example : Illustration for the factored framework

To illustrate the factored representational methodology, we will use the *Coffee Robot* example of a feature-based, stochastic, sequential decision problem, following the work of [26]. In this example, a robot, aka the learning agent, must go to a coffee shop to buy a cup of coffee for its owner who is located at its office. When it is raining, it must get an umbrella to stay dry when going to the shop. The state of the system is composed of six binary variables s_i where $Dom(s_i) = \{0, 1\}$ (corresponding respectively to False and True). These variables are :

* s_H : Has the owner a coffee ?
* s_C : Has the robot a coffee ?
* s_W : Is the robot wet ?
* s_R : Is it raining ?
* s_U : Has the robot an umbrella ?
* s_O : Is the robot in the office ?

This problem being composed of 6 binary variables, there is $2^6 = 64$ possible states. In this problem, four actions are available to the robot :

* *Go* : Move to the other location;
* *BuyC* : Buy a coffee (only available in the coffee shop);
* *DelC* : Deliver coffee to the owner (only possible in the office and if the robot has coffee);
* *GetU* : Get an umbrella (only in the office).

Actions can be noisy to represent stochastic problems. For instance, when the robot gives the coffee, its owner will get his coffee only with a given probability (the cup may fall). Thus, when the action DelC is executed in the state $s = (s_C = 0, s_H = 1, s_W = 0, s_R = 1, s_U = 0, s_O = 1)$ (the robot is in the office and the owner does not have a coffee), the transition function of the problem defines :

— $\mathcal{P}((s_C = 1, s_H = 1, s_W = 0, s_R = 1, s_U = 0, s_O = 1)|s, DelC)$ = 0.8;

— $\mathcal{P}((s_C = 0, s_H = 1, s_W = 0, s_R = 1, s_U = 0, s_O = 1)|s, DelC)$ = 0.2.

Otherwise all remains deterministic. Finally, for the reward function, the robot gets a reward of 0.9 when the owner has a coffee (0 when it does not) and 0.1 when it is dry (and 0 when the robot is wet). The reward the robot obtains when the owner gets a coffee is larger than the reward obtained when the robot is dry so as to specify that the task of getting a coffee has a higher priority than the constraint of staying dry.

6.1.2 Formalism

The idea of factorisation comes from the necessity to represent large-scale problems that cannot be solved with the standard MDP representations. FMDP [26] exploit the structure of the problem to represent large MDPs compactly when the state of the problem can be decomposed into a set of random variables. It is a feature-based representation of MDP framework in which the set of of possible states can be characterised by a set of random variables. Formally, the set of possible states S is described by a set of random variables $S = S_1, \ldots, S_N$ where each variable S_i can take different values in its domain $Dom(S_i)$. The environment state s can be characterised by a finite set of random variables $s = \{s_1, \ldots, s_N\}$.

It uses dynamic Bayesian networks (DBNs) to represent the transition probabilities associated with a specific action a. The Bayesian network nodes correspond to local state variables s_i and is partitioned into two sets : the state of the system at time t before the action is performed, s_i^t and the state after the action is executed at time $t+1$, s_i^{t+1}. Note that we will use indifferently s' and s^{t+1} to represent the state value at time $t+1$. Edges between these two set of nodes represent direct probabilistic influence among the corresponding variables under the action a. We denote by $\mathcal{G}_a$ the DBN under action a which is a two-layer directed acyclic graph whose nodes are $\{s_1, \ldots, s_N; s'_1, \ldots, s'_N\}$. Figure 6.1 is an illustration of the DBN associated with action $DelC$ in the Coffee Robot example. The action 'Deliver Coffee' will provide direct influences only in local variables that are involved with this action, namely s_H, s_C and s_O, i.e. if the human has the coffee, if the robot has the coffee and if the robot is in the office.

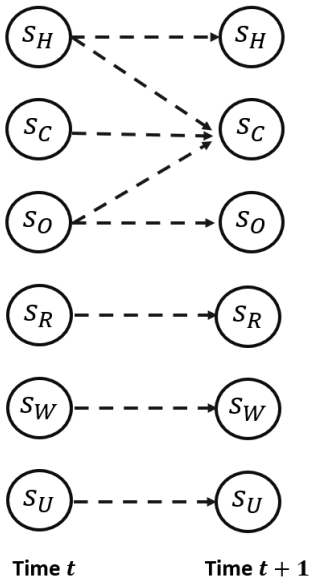


FIGURE 6.1 – Example of the DBN under action $DelC$ in the Coffee Robot

For a DBN under action a we can extract the conditional probability distributions (CPDs) or conditional probability tables (CPTs) of each variable s_i at time $t+1$ and under an action a, denoted $P_i(s'_i|Pa(s_i), a)$, where $Pa(s_i)$ represents the parents variables of s_i at time t. Table 6.1 represents the CPT of variable s_C under the action $DelC$. Note that the entries of the CPT are extracted from the DBN of Figure 6.1, i.e. that the parents of variable s_C under action $DelC$ are $\{s_H, s_C, s_0\}$. The left part expresses the values of parents variable at time t while the right part expresses the probability that the variable $s_C = 1$ at time $t+1$.

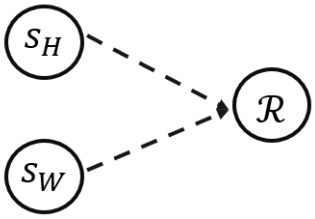


FIGURE 6.2 – Representation of the reward function $\mathcal{R}$ in the Coffee Robot

s_C^t	s_H^t	s_O^t	s_C^{t+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0.8
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

TABLE 6.1 – CPT of local variable s_C under action $DelC$

Now, we can express the global conditional distribution $P(s'|s,a)$ as a product of local conditional probabilities. We need for this purpose to ensure that there exists no synchronic arcs at time $t+1$.

Proposition 4 ([100]). *If* $\forall s'_i, s'_j \in \mathcal{S}', s'_i \perp s'_j | s$, *then*

$$P(s'|s,a) = \prod_i P_i(s'_i|Pa(s_i),a)$$

For a given function and a given context, it is not necessarily required to test every variable on which the function depends to define the output of the function. Such property is named context-specific independence.

Definition 11 (Context [2]). A context $c \in Dom(C)$ is an instantiation of a multivariate random variable $C = (C_0, \ldots, C_K)$ such that $C \subseteq S$. In other words, the context c is the instantiation of some state variables s_i and will mainly be used for parents-child relations.

Finally, notice that the MDP elements can also be factored such as the probability with local CPTs. It is the case for the reward function for example that can be linearly decomposed. We display in Figure 6.2 the dependencies between state variables and the reward $\mathcal{R}$. Only local variables s_H and s_W have influence on the reward. Table 6.2 shows the reward values given s_H and s_W.

s_H	s_W	$\mathcal{R}$
0	0	0.1
0	1	0
1	0	1
1	1	0.9

TABLE 6.2 – Tabular representation of the reward function in the Coffee Robot

Naturally, we can decompose linearly the reward function such that $\mathcal{R}(s) = \mathcal{R}(s_H) + \mathcal{R}(s_w)$.

In a nutshell, the factored framework allows to represent more compactly the elements of the MDP by decomposition of the reward function, policy or value functions and the transitions. This is done by understanding the dependencies that occur between two time steps in the system under the decisions of the agent. However, this factored framework requires specific representational structure in order to do planning and learning algorithms.

6.1.3 Factored MDP methodologies

This section describes different planning methods to solve problems specified as FMDPs. Rather than describing the algorithms in details, we describe the different representations and data structures they use as an outline of their main properties. We first give an overview of techniques assuming the system's dynamics and reward are known and the agent has access to $\mathcal{P}$ and $\mathcal{R}$, therefore considering MDP scenarios. As we have seen, the aim of factored representation is to represent more compactly the environment dynamics and reward. This brings several benefits : smaller memory implementation for large-scale systems and higher speed of convergence with less computations. The first works consider graphical representations with decision trees and decision diagrams [26, 83, 2]. These two objects allow to represent relations between state variables and conditional probabilities in a compact form, but also all the agent's objects (value function, policy). FMDPs can be solved by Structured Dynamic Programming (SDP) algorithms [26], but also Linear Programming (LP) [50]. However we will only focus on SDP algorithms in this chapter.

6.1.3.1 Decision Tree representation

Decision trees can represent any function by partitioning its input space and associating the output value to each of these partitions. A decision tree is composed of :

* *internal or decision nodes* : they represent a test on a variable of the input space. They are parents of other nodes in the tree and define the partitions of the input space;
* *edges* : they connect a parent interior node to a child node and constrain the value of the variable tested at the parent node to one value to reach the child node;
* *external or leaves nodes* : they represent the terminal nodes of the tree and define the value of the function for the partition defined by the parent (internal) nodes.

A function f represented by a decision tree is noted $Tree[f]$. Graphically, we represent decision trees with the following convention : for an internal node testing a Boolean variable s_i, the left and right edges correspond respectively to $s_i = 1$ and $s_i = 0$ (or respectively s_i being *true* and s_i being *false*). We display in Figure 6.3 the tree representation for the CPT of Table 6.1. Whereas 8 lines are required (Table 6.1) for the tabular form, only 4 are required for the same function with a decision tree.

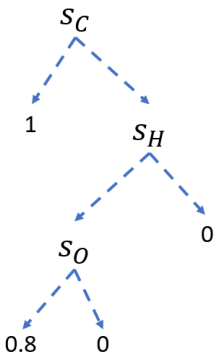


FIGURE 6.3 – Tree representation for the CPT of variable s_C under action $DelC$

Standard SDP algorithms such as Structured Value Iteration (SVI) and Structured Policy Iteration (SPI) use decision trees as factored representation. They use trees to represent functions of the FMDP, such as reward functions, transition functions, policies and value functions. The two algorithms SVI and SPI can be seen as an efficient way to perform the Bellman-backup operation on trees, expressed as follows :

$$Tree[Q_a] = Tree[R_a] + \gamma Tree[P_a V]$$

In order to be able to compute and represent the trees, SVI and SPI use several operations on decision trees : *merging trees*, *simplifying trees* and *appending trees*. These operations are mainly to compress the representations by aggregating nodes of the trees due to the factored structure and relations between variables. Consequently, rather than iterating on all the states of the problem to update the value function as Value Iteration and Policy Iteration do, SVI and SPI compute the update only for each leaf of the decision tree, decreasing the computation when states are aggregated and represented with one leaf.

6.1.3.2 Decision Diagram representation

An other SoTA technique to solve FMDP with SDP is the representation of the problem with *Algebraic Decision Diagrams* (ADD). ADDs are a generalisation of binary decision diagrams. An ADD is defined by :

* *internal or decision nodes* : they represent a test on a variable from the input space. They are the parent of two edges corresponding respectively to the values and;
* *edges* : they connect each parent internal node to a child node depending of its associated value or;
* *external or leaves nodes* they represent terminal nodes in the diagram and are associated with the value of the function in the subspace defined by the set of tests of the parent nodes to reach the leaf.

In comparison to decision trees, ADDs have several interesting properties. First, since an order is given, each distinct function (transitions, rewards, values, policy) has only one representation. Moreover, the size of the representation can be compressed because identical subgraphs can be factored in the description. be taken into account in the description. Finally, optimised algorithms have been proposed for most of the basic operators, such as multiplication, addition or maximisation of two ADDs (similar to tree operations). In addition, the ordering is exploited to handle ADDs more efficiently compared to decision trees where there is no ordering. Figure 6.4 shows the different representations of trees and ADDs.

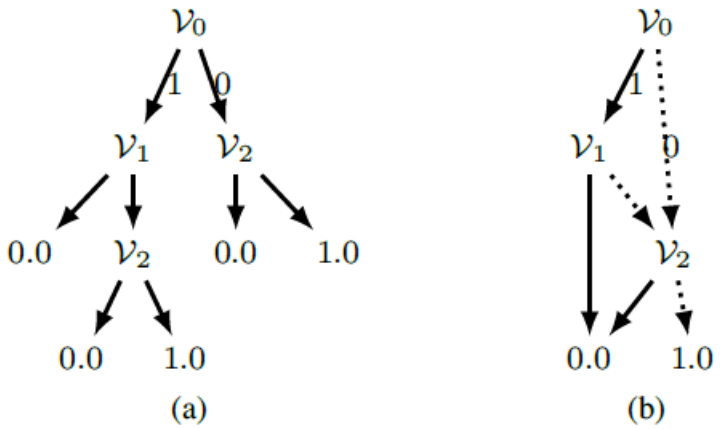


FIGURE 6.4 – Comparison of the representation of a function f as a decision tree Tree [f] (a) and as an algebraic decision diagram ADD[f] (b); Figure from [2]

Boutlilier et al. proposed SPUDD in [59], which stands for *Stochastic Planning Using Decision Diagrams*. It uses the ADDs instead of decision tree to represent the transition function of FMDPs. This results in a more compact representation and consequently it accelerates the computations. It is such as SVI, based on the Value Iteration algorithm, adapted with ADDs in scenarios where all state variables are binary. Both of the advantages (ordering, more compact) described above allow SPUDD to perform significantly better than SPI or SVI on most problems proposed in the FMDP literature [26] (Coffee Robot, Taxi problem, etc.).

Remark. The works with decisions trees and ADDs [26, 83] often show applications with binary feature variables in the state space. In this setting, Magnan [100] extended the current framework with multi-valued feature variables by treating more complex graphical representation of FMDPs. However, there also exists few works that considered factored tabular representation that can handle multi-valued environment variables.

6.1.3.3 Tabular representation

On the other hand, some works have been investigating the factored form of classic MDP algorithms in the initial representation, namely tabular. This includes Factored Policy Iteration [79] and Factored Value Iteration (FVI) [135]. In this work, we will focus on the FVI method that will be utilised in our Factored RL algorithm. To present the FVI algorithm, we first recall the update process in Value Iteration and in Approximate Value Iteration (AVI) that uses linear approximation of the value function V.

Value Iteration The Value Iteration algorithm [5] uses the Bellman equations (eq 3.1) to update the value function V. It starts with V_0 and update at iteration t the value function $V(s)$ for all state s with backprojection formula :

$$V_{t+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V_t(s')\right]$$

We can rewrite the set of equations in matricial form :

$$V_{t+1} \leftarrow \max_a (r^a + \gamma P^a V_t)$$

Approximate Value Iteration with Linear Approximation The Value Iteration algorithm also exist for linear approximation of the value function [4]. Consider the value function V to be a linear combination of K basis functions h_k where $K << |\mathcal{S}|$. We have :

$$V(s) = \sum_{k=1}^{K} w_k h_k(s)$$

By substituting V_t by $\mathcal{H}w_t$, we can rewrite the Bellman equation 3.1 in its matricial form by :

$$\mathcal{H}w_{t+1} \leftarrow \max_a(r^a + \gamma P^a \mathcal{H}w_t)$$

However the right-hand-side (r.h.s.) might not be contained in the image space of $\mathcal{H}$. Hence, we need to project the r.h.s. on the image space with a projector $\mathcal{G}$:

$$w_{t+1} \leftarrow \mathcal{G}\left[\max_a(r^a + \gamma P^a \mathcal{H}w_t)\right]$$

The choice of the projection operator is important for convergence and is discussed later in the factored value iteration part.

Factored Value Iteration Factored Value Iteration [135] presents a factored version of the Value Iteration with convergence guarantees. We first have linear decomposition of the reward and value functions.

$$R(s,a) = \sum_{j=1}^{J} R_j(s[Z_j], a)$$

$$V(s) = \sum_{k=1}^{K} h_k(s[C_k]) \cdot w_k$$

where $Z_j \subseteq \mathcal{S}$ and $C_k \subseteq \mathcal{S}$ are sets of local variables. Here, $s[Z_j]$ and $s[C_k]$ correspond respectively to the values of local variables in Z_j and C_k for state s. For example in the coffee robot, the reward function was decomposed linearly with two reward functions $\mathcal{R}_1$ and $\mathcal{R}_2$. In this setting, we would have $Z_1 = s_H$ and $Z_2 = s_W$ and $s[Z_1]$ would be the value that takes local variable s_H.

The idea of factored value iteration is to replace Bellman's equation with a factored version, by considering factored local transitions P_i with their CPTs representation and linear value function approximation. The linear decomposition gives us :

$$V_{t+1} = \sum_{k=1}^{K} h_k(s[C_k]) \cdot w_k^{t+1}$$

The factored transition probability under an action a and under the same scopes C_k used in the value function decomposition, gives us :

$$P(s'|s,a) = \prod_{i \in C_k} P_i(s'_i | Pa(s_i), a)$$

By replacing the terms in the original Bellman equation, we have :

$$\sum_{k=1}^{K} h_k(s[C_k]) \cdot w_k^{t+1} = \mathcal{G}\max_a \left[\sum_{j=1}^{J} R_j(s[Z_j], a)\right] + \gamma \sum_{k=1}^{K} \sum_{s'[C_k] \in \mathcal{S}[C_k]} \Big(\prod_{i \in C_k} P_i(s'_i | Pa(s_i), a)\Big) h_k(s[C_k]) \cdot w_k^t \tag{6.1}$$

We can observe in equation 6.1 that the update complexity is reduced compared to the original Bellman equation 3.1 since we do less calculations by considering the factored form of transitions. By considering the backprojection matrix $\mathcal{B}$ such that :

$$\mathcal{B}^a_{s,k} = \sum_{s'[C_k] \in \mathcal{S}[C_k]} \Big(\prod_{i \in C_k} P_i(s'_i | Pa(s_i), a) \Big) h_k(s'[C_k])$$

we have in matricial computation the update of the value function's weights by :

$$w^{t+1} = G \max_a \Big[r^a + \gamma \mathcal{B}^a w^t \Big]$$

where G is the matrix form of the projection operator $\mathcal{G}$.

Overall, they show that their algorithm is only polynomial in the number of basis functions k, which demonstrates a huge gain compared to original MDP Value Iteration. Also, the authors [135] propose in their work a *sampling* method to work on a subset of the original state space $\hat{\mathcal{S}} \subseteq \mathcal{S}$ where $|\hat{\mathcal{S}}| = \text{poly}(N)$. Algorithm 28 displays the FVI algorithm.

Algorithm 28: Factored Value Iteration

Input: Basis functions H_i, G projector, $V^0 = Hw^0$, q^0, ξ accuracy
Output: q^*, V^*
Data: Statistics and DBNs known

N_1 number of samples // Sampling method
$\hat{S}$ uniform random N_1-elements subset of S
Create $\hat{H}$ and $\hat{G}$
Create $\hat{B}^a = P^{\hat{a}} H$ and $\hat{r}^a$ for all actions $a \in \mathcal{A}$
$w^0 = 0, t = 0$
repeat
    $w^{t+1} = G \max_a \Big[r^a + \gamma \mathcal{B}^a w^t \Big]$
    $\Delta_t = ||w^{t+1} - w^t||$
    $t \leftarrow t + 1$
until $\Delta_t \leq \xi$
return w^t, V^t, q^t

6.1.4 Factored Reinforcement Learning

So far, we have provided an overview of factored techniques to solve Markov Decision Process scenarios where the agent was provided the true environment model. These methods are inspired by DP algorithms often used for MDP resolutions. Several works have also proposed factored solutions in the RL paradigm where the agent has missing knowledge and needs to learn from data the model of the world to perform planning. We find in these proposal the same characteristics as in the model-based RL domain, namely, the learning and planning in the factored framework. *Factored Reinforcement Learning (FRL)* [73, 83] is a model-based Reinforcement Learning approach to FMDPs where the transition and reward functions of the problem are learned. First, it uses the methods of Supervised Learning to construct the factored representation of the environment. In parallel, planning algorithms can use this representation to build the policy.

The FRL algorithms are mainly inspired by Dyna architectures (deterministic models) and MDP online techniques (stochastic models). Among these methods the structural knowledge is an hypothesis that can be assumed

or not. Therefore, some works [73, 83] require to learn only the statistics because the agent already knows how the environment variables influence each other. In other words, the learning agent is provided the relational structure of the environment but does not know the statistics. Nevertheless, other researchers [35] have tempted to consider more complex scenarios where the agent should also learn the relations between variables from the collected data.

6.1.4.1 Factored Dyna architectures : SDYNA

Structured-Dyna (SDYNA) framework proposed by Degris et al. [35], is a structured version of the DYNA architecture. SDYNA integrates incremental planning algorithms based on FMDPs with supervised learning techniques building structured representations of the problem. It uses the decision tree representation. The inner loop of SDYNA is decomposed into three phases :

* *Acting* : choosing an action according to the current policy, including some exploration ;
* *Model-Learning* : updating the model of the transition and reward functions of the FMDP from (s, a, r, s') observations ;
* *Planning* : updating the value function $Tree[V]$ and policy $Tree[q]$ using one iteration of SDP algorithms.

6.1.4.2 Factored MDP online

It has also been proposed solutions for stochastic models methods. For model-based Reinforcement Learning techniques, Kearns and al. present in [73] a factored implementation of the E_3 MDP-online algorithm, called *DBN-E3*. The authors proposed a provably efficient and near-optimal algorithm that generalises the E3 technique. In their work, they assume that they are given both an algorithm for approximate planning, and the graphical structure (but not the parameters) of the DBN. Unlike the original E3 algorithm, this factored version of E3 exploits the DBN structure to achieve a running time that scales polynomially in the number of parameters of the DBN, which may be exponentially smaller than the number of global states.

Guestrin et al. [51] also proposed a factored version for the R-max algorithm, called *Factored R-max*. In their work, Guestrin et al. address a significant shortcoming of Factored E3 : namely that it requires an oracle planner that cannot be feasibly implemented. They propose an alternative approach that uses a practical approximate planner and approximate linear programming. Furthermore, they develop an exploration strategy that is targeted toward improving the performance of the linear programming algorithm, rather than an oracle planner. This leads to a simple exploration strategy that visits the relevant states for the LP solution, and achieves logarithmic sample efficiency in the size of the problem description. Moreover, they show in experimental results that their proposal performs better.

Finnaly, Strehl et al. [131] have studied the optimality and performances of both factored E3 and factored Rmax. They have shown that factored Rmax was performing near-optimally on all but a number of timesteps that is polynomial in the size of the compact representation, which is often exponentially smaller than the number of states and showed the equivalence with the result obtained by Kearns and Koller [73] for their DBN-E3 algorithm. Moreover they proposed a new algorithm inspired from the two above, *Factored IE* that uses the Interval Estimation approach to exploration and can be expected to outperform factored Rmax on most domains.

6.1.4.3 Learning the structure - SPITI

Finally, some works have considered cases where the learning agent was not provided the structure of the environment (e.g. DBN). Degris et al. [35] have extended the SDYNA algorithm in scenarios where the agent also needs to learn the structure of the FMDP and derived the *SPITI*. SPITI is a particular instance of SDYNA using ϵ-greedy as exploration method, the Incremental Tree Induction (ITI) algorithm to learn the model of transitions

and reward functions as a collection of decision trees. During the learning phase of the SPITI algorithm, the decision trees are built from the flow of examples, extracted from the current state of the environment. Each tree $Tree[P]$ quantifying the probabilities of transitions is built directly without building the corresponding DBN for each action.

Last, Strehl et al. [132] also studied this scenario and extend existing algorithms with learning of conditional probability tables of DBNs in cases in which DBN structure is not known in advance. Their method learns the DBN structures as part of the reinforcement-learning process and provably provides an efficient learning algorithm when combined with factored Rmax.

6.1.5 Feasibility study of FVI in the Coffee Robot Problem

Before presenting our factored RL algorithm and the factored representation of the multi-tier Cloud architecture, we first did a feasibility study of the convergence of FVI on the Coffee Robot example since we will use it for the planning phase.

We implemented FVI and compared the results with classical VI algorithm and demonstrated that the two policies were equal. The value functions were different due to linear approximation in the FVI method. Note that the value function of FVI depends on the basis functions choices that have been selected here to integrate the reward function knowledge. We display the policy and value function values for some states $s \in S$ in Table 6.3. We can remark that the value functions of both algorithms have similar trend with a gap of ~ 30 between them. This gap might be due to a missing basis function that could integrate more knowledge, e.g. a basis function that takes as input both s_H and s_W or considering polynomial features.

States	VI q	VI exact V	FVI q	FVI approximate V
$(s_C = 1, s_H = 0, s_W = 0, s_R = 1, s_U = 0, s_O = 1)$	DelC	99.79	DelC	60,93
$(s_C = 0, s_H = 0, s_W = 0, s_R = 1, s_U = 0, s_O = 1)$	GetU	99,99	GetU	61,89
$(s_C = 0, s_H = 0, s_W = 0, s_R = 1, s_U = 1, s_O = 1)$	Go	99,99	Go	61,89
$(s_C = 0, s_H = 0, s_W = 0, s_R = 0, s_U = 0, s_O = 0)$	BuyC	99,99	BuyC	61,89
$(s_C = 1, s_H = 0, s_W = 0, s_R = 0, s_U = 0, s_O = 0)$	Go	100,98	Go	63,15

TABLE 6.3 – Policies and value functions returned by VI and FVI for some Coffee Robot states

6.2 Factored Model-Based Reinforcement Learning

We describe in this section our proposal for a factored model-based reinforcement learning algorithm. As opposed to principal literature solutions with Decision Trees and Decision Diagrams, we work here with probability tables such as in DBN-E3 or Factored R-max. Our method is inspired by MDP online techniques where the autonomous agent learns the dynamics model $\mathcal{P}$ of the world by interactions before performing planning. The only variation is that here we operate with a factored representation to accelerate the convergence by reducing the calculations. We propose **FMDP online** (Algorithm 29) that uses methodologies from MDP online algorithms and the Factored Value Iteration (FVI) [135] for the planning phase. Recall that the DBN-E3 was assuming in its proposal that a planning method was provided for their algorithm, yet not treated in their paper [73]. Here, we consider that the planning method is the FVI algorithm and the learning agent will learn the factored model of the environment to plan in a factored manner.

6.2.1 Factored probabilities inference

As opposed to usual model-based methods, we now assume that the agent has a supplementary knowledge. It understands the relational structure of the environment : how state variables are related to each other. The dynamics are now represented by DBNs and local conditional distributions $P_i(s'_i|Pa(s_i), a)$, implemented with conditional probability tables (CPTs). However the agent does not know the statistics and needs to interact with the environment to learn these statistics for planning. We assume that the reward function is known and only dynamics P_i have to be updated.

As in R-max [27], the method starts by assuming that the initial system is in an absorbing state, i.e. we initialise all local CPTs as identity matrices. This is a fair assumption at the beginning since the agent does not know how the environment behaves. At the end of an episode e, it updates its local CPTs from its experiences. The update is done by counting occurrences, i.e. :

$$P(s'_i|Pa(s_i), a) = \frac{\sharp(s'_i, Pa(s_i), a)}{\sharp(Pa(s_i), a)}$$

where $\sharp(\cdot)$ denotes the number of occurrences and tuple $(s'_i, Pa(s_i), a) = (s'[i], s[Pa(s_i)], a)$. As a difference with R-max and E3 algorithms presented in Section 5.1, the method here assumes that the agent does know the full state space and does not need to iteratively solve known sub-MDPs plus an absorbing unknown MDP like in E3. Here if the model is approximated, it can be solved with dynamic programming methods in the whole environment. Once the local CPTs are updated from experiences, the agent can perform planning with its learned model $\tilde{\mathcal{M}}$.

6.2.2 Factored planning

The algorithm is divided in two steps : statistical inference to update the factored dynamics then planning to update the value function. The planning is done in a factored manner with the Factored Value Iteration algorithm [135]. We remind here that the value function V is decomposed with a linear approximation $V(s) = \sum_{k=1}^{K} w_k h_k(s)$, thus the planning step ought to update the weights w_k of the linear decomposition. However, one major drawback of model-based reinforcement learning techniques is the planning step with a poor model approximation that leads to a policy with poor performance. To overcome this problem, since the agent might not be confident in the approximated model $\tilde{\mathcal{M}}$ during first episodes, we only run FVI algorithm for a few number of iterations. In practice, we do e number of iterations to update the weights w of the value function V, where e is the current running episode number. This means that the learning agent will do more planning iterations as time goes by : few iterations at the beginning and many iterations at the end.

6.2.3 Exploration-Exploitation trade-off

To explore and exploit efficiently, we provide the agent with an ϵ-greedy policy. This choice was made arbitrarily regarding the literature techniques. Some algorithms often consider this technique to treat this exploration-exploitation dilemma, however other researchers propose that the agent follows its policy q and provide incentive reward for exploration (such as in Rmax).

6.2.4 Inputs and parameters

The inputs of the FMDP online algorithm are mainly inputs of the FVI planning method. It includes the choice of 'good' basis functions and the choice of the projection operator. We will describe in the next section our

choices for the application of our method in the tandem queue MDP model. Finally, we display in Algorithm 29 the whole method.

Algorithm 29: Factored MDP online algorithm

Input: Basis functions H_i, G projector, local CPTs $P_i = \mathbf{I}$, $V^0 = Hw^0, q^0$
Output: q^*, V^*
Data: Statistics unknown, DBNs known

```
for e ∈ MaxEpisode do
    for i ∈ MaxIterations do
        Select state s ∈ S
        Take action a ∈ A with epsilon-greedy policy
        Observe s' and reward r(s, a) and collect tuple < s, a, r, s' >
    /* Factored inference end of the episode                    */
    for i = 1, .., n do
        Update local transitions P(s'_i|Pa(s_i), a) → approximated model FACT(M̃)
    /* Factored planning                                        */
    w^e, q^e = Factored Value Iteration planning with e update iterations
```

6.3 Factored MDP representation of the Tandem Queue Environment

In this section we first present the factored representation of the tandem queue environment regarding state space, dynamics and reward. Next we provide the inputs design of our factored Reinforcement Learning method for this given application.

6.3.1 Factored model representation

Understanding structural properties of the tandem queue system is key to help the agent learn faster the auto-scaling policy. We model the tandem queue environment with a factored approach to bring this knowledge to the learning agent.

6.3.1.1 Original Tandem Queue MDP with DBNs

To build the factored representation of the current tandem queue MDP, we need to define the relationship between all state variables from time t to time $t+1$. This is depicted with a dictionary of child-parent variables in the Table 6.4.

Child variables at time t+1	Parents variables at time t
m_1	m_1, k_1
m_2	m_1, k_1, m_2, k_2
k_1	k_1
k_2	k_2

TABLE 6.4 – Child-parent architecture

In order to see how we can factorised the MDP, we need to represent it with Dynamic Bayesian Network. In factored approaches, it is common to have a DBN for each action $a \in \mathcal{A}$. Yet all the DBNs in the tandem queue scenario have the same structure (see Figure 6.5). Now if we currently try to use factorisation on the current state space representation, then we will have dependencies within the state at time $t+1$, which will violate Proposition 4 and does not allow us to use the factored representation. In other words, the factored form of the probability distribution $P(s'|s,a) = \prod_{i=1}^{N} P_i(s'_i|Pa(s_i),a)$ is not possible because the two events, departure from node 1 ($m_1 \rightarrow m_1 - 1$) and arrival in node 2 ($m_2 \rightarrow m_2 + 1$), are the same, thus interdependent at time $t+1$. In details, when a customer goes from node 1 to node 2, we have :

$$p(s'|s,a) \neq p(m'_1|m_1,k_1,a)p(m'_2|m_1,k_1,m_2,k_2)p(k'_1|k_1,a)p(k'_2|k_2,a)$$

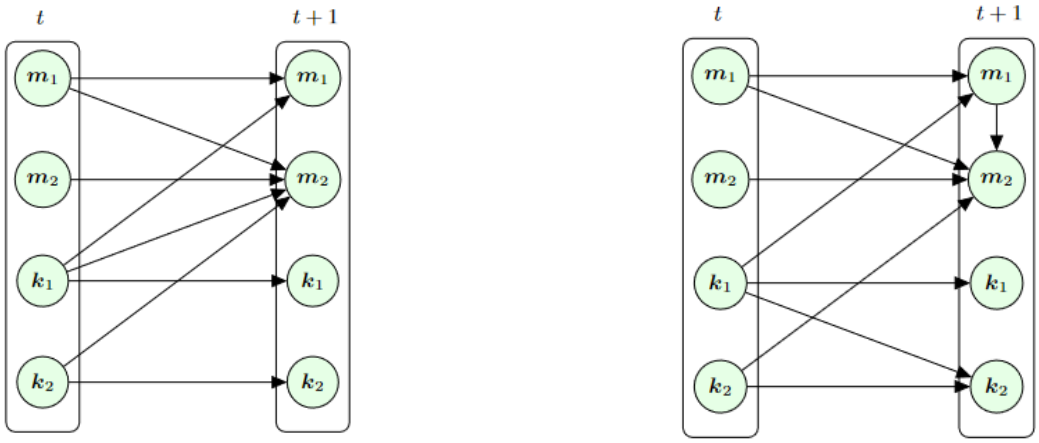


FIGURE 6.5 – Dynamic Bayesian Networks under any action a for original tandem queue MDP

This dependency between variable m_1^{t+1} at time $t+1$ and m_2^{t+1} is demonstrated in Figure 6.6 which provides Pearson correlations between all state variables at time t and time $t+1$ over 20000 samples collected by interactions with the environment. We can see that $corr(m_1^{t+1}, m_2^{t+1}) = -0.31$ showing negative correlation.

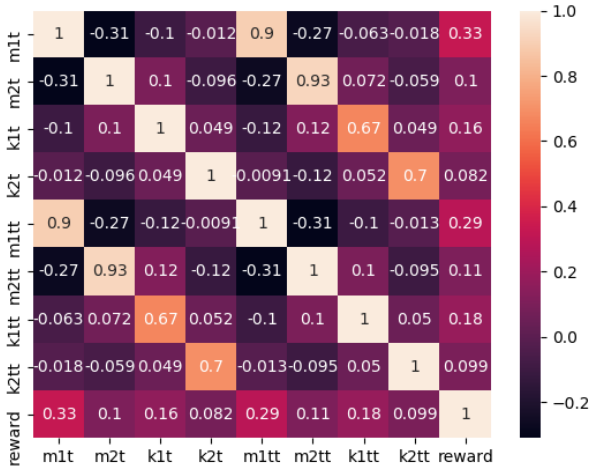


FIGURE 6.6 – Correlation between state variables after 20000 samples collected by RL agent

6.3.2 Factored MDP model with Augmented State Space

We did slight modification of the tandem queue MDP model to build the factored model. To overcome the violated assumption, we extended the state space to break the dependency between the variables m_1 and m_2 at time $t+1$. This is done while considering variables m_1 and m_2 at previous time step $t-1$, denoted m_1^p, m_2^p and the current variable m_1 at time t. We now consider the state at time t : $s = (m_1^p, m_1, m_2^p, k_1, k_2)$. We draw the DBN associated to the new representation of the state space in Figure 6.7.

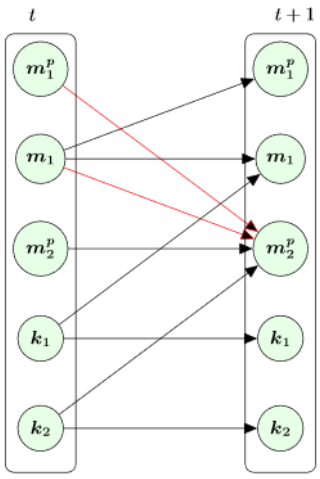


FIGURE 6.7 – Dynamic Bayesian Network representation for factored tandem queue MDP

We used the *pyAgrum* library [37] for implementation of CPTs and DBNs. We show in Figure 6.11 an example of the local CPT of variable k_1 under all actions a, where the CPT structure is derived from the DBN.

	k1tt				
k1t	0	1	2	3	4
0	1.0000	0.0000	0.0000	0.0000	0.0000
1	1.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	1.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	1.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	1.0000	0.0000

FIGURE 6.8 – Action $a_1 = -1$

	k1tt				
k1t	0	1	2	3	4
0	1.0000	0.0000	0.0000	0.0000	0.0000
1	0.0000	1.0000	0.0000	0.0000	0.0000
2	0.0000	0.0000	1.0000	0.0000	0.0000
3	0.0000	0.0000	0.0000	1.0000	0.0000
4	0.0000	0.0000	0.0000	0.0000	1.0000

FIGURE 6.9 – Action $a_1 = 0$

	k1tt				
k1t	0	1	2	3	4
0	0.0000	1.0000	0.0000	0.0000	0.0000
1	0.0000	0.0000	1.0000	0.0000	0.0000
2	0.0000	0.0000	0.0000	1.0000	0.0000
3	0.0000	0.0000	0.0000	0.0000	1.0000
4	0.0000	0.0000	0.0000	0.0000	1.0000

FIGURE 6.10 – Action $a_1 = 1$

FIGURE 6.11 – Factored probabilities for variable k_1 under all actions a_1

6.3.2.1 Factored system's dynamics

We can henceforth build the factored dynamics of the tandem queue environment. We define the factored transition probabilities P_i for each local state variable s_i. First we have deterministic local transitions for servers variables k_1 and k_2 since these values only depend on the agent's decision a. Therefore, we will move from local states k_1 to $N(k_1 + a_1)$ and from k_2 to $N(k_2 + a_2)$ with probability 1 (see Figure 6.11).

We have the same factored transition for local variable m_1^p since m_1^p at time $t+1$ simply retrieves the value of local variable m_1 at time t. Therefore, the randomness in the system is only integrated in the factored transitions of local variables m_1 and m_2^p, which will represent possible events (arrival in node 1, departure from node 1 to node 2 and departure from node 2).

Now the full probability distribution can be written under the factored form.

$$\begin{aligned} P(s|s',a) \quad = \quad & P(m_1^{p'}|Pa(m_1^p),a) \cdot P(m_1'|Pa(m_1),a) \cdot \\ & P(m_2^{p'}|Pa(m_2^p),a) \cdot P(k_2'|Pa(k_2),a) \cdot \\ & P(k_1'|Pa(k_1),a) \end{aligned}$$

Therefore, we will use these local conditional probabilities in the factored Bellman equation 6.1 in the FVI planning phase of our FRL algorithm. This will avoid to do the computations for all states s and s' thus providing a huge gain in the number of computations.

6.3.2.2 Factored system's costs

The reward function is also linearly decomposed such as in the coffee robot example, i.e. that the learning agent has a linear decomposition representation of the reward for the planning phase. We decide naturally to represent two features, one for each network node i. We have $\mathcal{R}(s,a) = \mathcal{R}_1(m_1,k_1,k_2,a) + \mathcal{R}_2(m_1,m_2,k_1,k_2,a)$ where each $\mathcal{R}_i$ takes into account the local costs (activation, deactivation, service, holding, reject) in each node.

Therefore we have :

$$\begin{aligned} \mathcal{R}_1(m_1,k_1,k_2,a) \quad = \quad & (m_1 C_{H_1} + (k_1+a_1)C_{S_1})/\bar{\Lambda} \\ & + C_{R_1}\lambda\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma} \\ & + C_{A_1}\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma}1_{(a_1=1)} \\ & + C_{D_1}\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma}1_{(a_1=-1)} \end{aligned}$$

$$\begin{aligned}\mathcal{R}_2(m_1, m_2, k_1, k_2, a) &= (m_2 C_{H_2} + (k_2 + a_2) C_{S_2})/\bar{\Lambda} \\ &+ C_{R_2} k_1 \mu_1 \frac{\Lambda(s,a) + \gamma}{\bar{\Lambda} + \gamma} \\ &+ C_{A_2} \frac{\Lambda(s,a) + \gamma}{\bar{\Lambda} + \gamma} 1_{(a_2 = 1)} \\ &+ C_{D_2} \frac{\Lambda(s,a) + \gamma}{\bar{\Lambda} + \gamma} 1_{(a_2 = -1)}\end{aligned}$$

6.3.3 Objective function

We consider here a continuous time discounted model. The discount factor is denoted by γ. We define a Markov Deterministic stationary policy q as a mapping from state space to action space $q : \mathcal{S} \to \mathcal{A}$. This mapping defines the action to be performed in a given state s. From discounted model in [5], we define the stage cost function $\mathcal{R}$ as :

$$\mathcal{R}(s,a) = \frac{1}{\Lambda(s,a) + \gamma} \left[c_1(s,a) + c_2(s,a) \right] + h_1(s,a) + h_2(s,a).$$

We search the best policy q to minimise the expected discounted expected reward, then the objective function is :

$$V^*(s) = \min_q \mathrm{E}^q \left[\sum_{k=0}^{\infty} \exp^{-\gamma t_k} \mathcal{R}(s_k, q(s_k)) \mid s_0 = s \right], \tag{6.2}$$

with t_k the epoch and s_k the state of the kth transition.

6.3.4 Parameters selection of FMDP online

Also for the FVI algorithm we have a linear decomposition of the value function. Thus we need to define the basis functions which is very tough to adjust for good results. Last, we need to choose a projector for the factored Bellman update to compute the weights w. We describe our choices for basis functions and projection operator. We choose least-squares (L2)-projection for the projection operator $\mathcal{G}$, following recommendation from [135]. Least-squares fitting is very often applied for projecting value functions. In this case, the linear weights w are chosen so that it minimises the least-squares error $w = \mathrm{argmin}_w ||Hw - v||_2^2$. This corresponds to the linear projection $G = H^+$ (i.e., $w = H^+ v$) where $H^+ = H^T (H H^T)^{-1}$ is the Moore-Penrose pseudoinverse.

For basis functions selection, we decided after several tests, to select several sub-parts of the reward function. We have chosen 8 basis functions to combine as much as possible the relations between the local variables and the impact of each local variables such that it will be integrated in the value function.

Basis H	Functions
$h_1(m_1^p)$	$(C_{h_1} \cdot m_1^p)/\bar{\Lambda}$
$h_2(m_2^p)$	$(C_{h_2} \cdot m_2^p)/\bar{\Lambda}$
$h_3(k_1)$	$(C_{s_1} \cdot k_1)/\bar{\Lambda}$
$h_4(k_2)$	$(C_{s_2} \cdot k_2)/\bar{\Lambda}$
$h_5(m_1^p, k_1, k_2)$	$\lambda C_{r_1} \cdot 1_{(m_1^p = B_1)}/\bar{\Lambda}$
$h_6(m_1^p, m_2^p, k_1, k_2)$	$\min(m_1^p, k_1) \mu_1 C_{r_2} \cdot 1_{(m_2^p = B_2)}/\bar{\Lambda}$
$h_7(m_1^p, k_1)$	$\big((C_{h_1} \cdot m_1)^2 + C_{s_1} \cdot k_1\big)/\bar{\Lambda}$
$h_8(m_2^p, k_2)$	$\big((C_{h_1} \cdot m_1)^2 + C_{s_1} \cdot k_1\big)/\bar{\Lambda}$

TABLE 6.5 – Basis functions selection for factored value iteration

6.4 Numerical Results

We present in this section the comparison between our method **FMDP online** and state-of-the-art Reinforcement Learning algorithms on the tandem queue system.

6.4.1 Environments and Simulation Parameters

We keep the same python Gym simulator as described in Chapter V Section 5.4. What changes now is how the RL agent perceives the environment. In this factored approach, it has a new observation of the environment as described above $s = (m_1^p, m_1, m_2^p, k_1, k_2)$.

6.4.1.1 Cloud parameters

We first describe cloud simulations parameters. We consider two cloud scenarios of different scales $C_1 : \{B_1 = B_2 = 10, K_1 = K_2 = 3, \mu_1 = \mu_2 = 2, \lambda = 8\}$ and $C_2 : \{B_1 = B_2 = 15, K_1 = K_2 = 5, \mu_1 = \mu_2 = 2, \lambda = 15\}$. For all these scenarios the costs parameters are the same in the two nodes : $\{C_a = 1, C_d = 1, C_s = 2, C_h = 2, C_r = 10\}$.

6.4.1.2 Algorithms comparison

We compare our **FMDP online** method with state-of-the-art RL algorithms. We choose one model-free RL method : **Q-Learning**, one buffer-based RL method : **Dyna Q** and finally the conventional model-based RL method : **MDP online**. The MDP online algorithm works similarly to FMDP online. However it works directly with the global conditional distribution $P(s'|s, a)$. Therefore it can suffer from heavy update computations because of the matrix size in large-scale systems. This is one issue our method can resolve since it only updates small size matrices (local conditional probabilities P_i), and does not require excessive amount of memory for problems in large-scale systems. We also recall that in the FVI algorithm we can use a sampling method that considers only a subset $\mathcal{S}' \subseteq \mathcal{S}$ of the whole state space.

6.4.1.3 Simulation and algorithms parameters

We detail here the simulation and algorithms parameters. Learning phase runs for 100 episodes of length 10000 iterations. One iteration corresponds to a transition from state s with action a to state s' with reward r. The agent runs epsilon-greedy policy over the whole learning process, with initial epsilon set to $\epsilon = 1$. We decrease epsilon value after each episode with a decay rate $\epsilon_{dec} = 0.99$. The discount rate is set to $\gamma = 0.9$.

6.4.2 Comparison Criteria between Algorithms

We compare the different algorithms during the learning phase. Performance measurements during learning are represented by a learning curve. We look at running time of algorithms and the average reward obtained after each episode to compare the goodness of the RL methods. We note that RL algorithms should also be assessed offline, e.g. with Monte-Carlo offline policy evaluation. For this purpose we need to convert the factored policy of the augmented state space for fair comparison with the classical policy. This is currently being investigated as part of further research. Henceforth we only display the comparison during the learning process.

FIGURE 6.12 – Average reward per episode over learning episodes in cloud C_1

6.4.3 Results

We display the average reward over learning episodes for first cloud scenario C_1 in Figure 6.12. We observe that our method **FMDP online** converges much faster than model-free Q-Learning and Dyna Q techniques. Its convergence is on par with the MDP online techniques. The model-based techniques with stochastic models require much less interactions with the environment to find optimal solution. To fully observe this convergence gain, we draw a black dot line on the figures that correspond to a discounted expected reward value. This helps the reader to see how fast algorithms can reach this mean discounted reward. For example in Cloud scenario C_1, the Q-Learning algorithm reaches this bound at the 100th episode while model-based techniques reach it much faster.

We display numerical results in Figure 6.13 for the second cloud scenario C_2. The learning process lasted for 100 episodes and the epsilon decay rate was smaller $\epsilon_{dec} = 0.95$. The same conclusions as in scenario C_1 are drawn. The number of iterations required to reach an average reward of 2.05 is impressively less for model-based RL techniques compared to model-free.

In Figure 6.14, we show the comparison of running time between the MDP online method and our FMDP online method for the two scenarios. In the first scenario, our method has a similar running time to the MDP online method, as shown in 6.14(a). As the problem scale grows, our method has a lower running time in the second scenario, as shown in 6.14(b). This demonstrates an advantage of factored solution in execution time with similar policy quality as the problem scale grows.

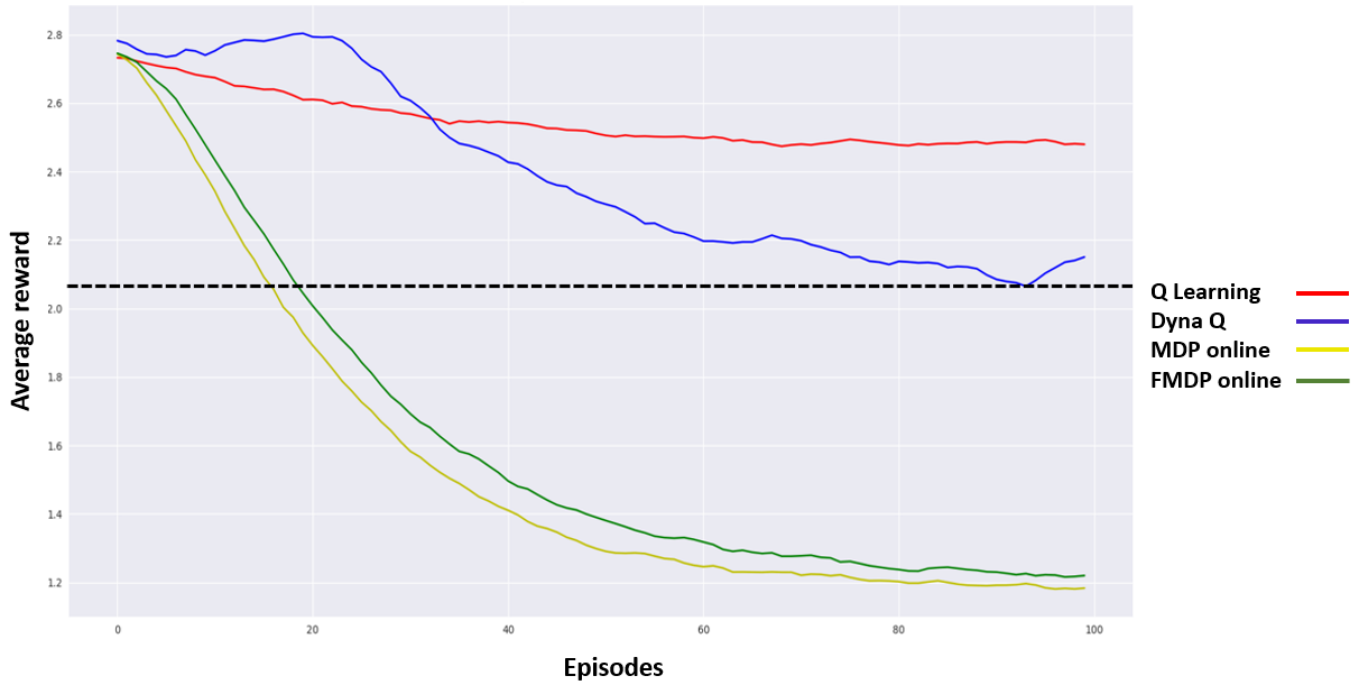


FIGURE 6.13 – Average reward per episode over learning episodes in cloud C_2

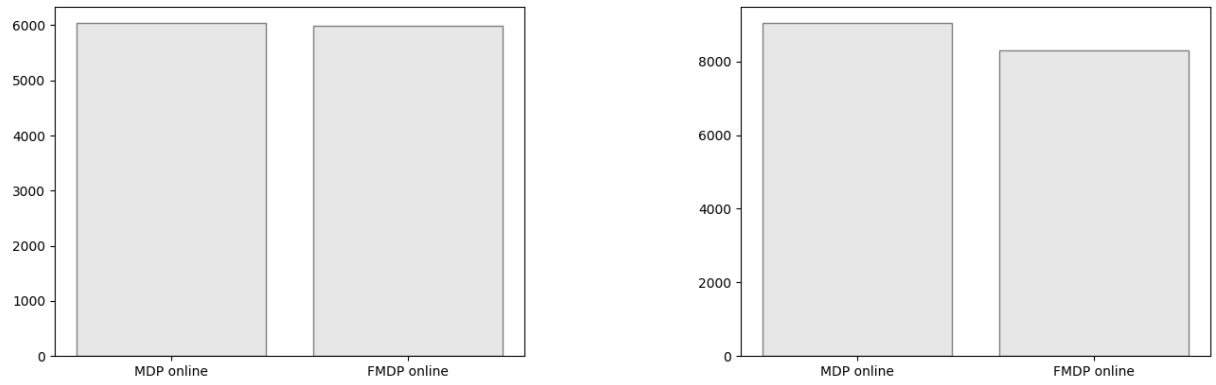


FIGURE 6.14 – Barplots showing running time of model-based RL algorithms

Last we consider a large scale cloud scenario $B = 50$ $k = 10$ with state space considering 250000 states. In this setting, it's not feasible for our computational setup to hold in memory the representations of the full transition matrix (25000x25000) and state values. Thus under tabular forms, only the FMDP online method is feasible, since it only requires lower dimensional arrays to represent local conditional distributions and uses sampling technique.

In order to fully grasp the gain of FMDP online technique, we provide simulations not only for a given number of episodes and iterations per episode, but for a same computation time.

6.5 Summary of the chapter

In this chapter we presented **FMDP online**, a factored Reinforcement Learning algorithm for optimal resource management in multi-tier Cloud architecture. We demonstrate that if we integrate local relations between state variables in the representation of the environment, it can help to overcome the 'curse of dimensionality' issues in standard RL methods.

We have shown the performance improvement obtained by the proposed method in a tandem queue system. It presents a good trade-off between policy quality and computation time. The model-free techniques such as Q-Learning or Dyna architectures may require too many iterations to converge. The model-based algorithms may require fewer iterations to converge. But they suffer greatly on large-scale systems simply due to an exponential increase of the state space. Working with huge matrix representation for the transition probability leads to long computations for the updates and low confidence in the approximated dynamics. Moreover as the state space grows, it may become infeasible to hold the transition matrix in memory. On the contrary, the factored approach have smaller size matrices for the local conditional distributions and can be quickly updated. The sampling technique of factored value iteration also allows to reduce the complexity by considering a subset of the whole state space.

Nevertheless this work requires further investigations. The first one is about evaluating learned policy offline with Monte-Carlo evaluations where we need to convert factored policy for fair comparison. Secondly we would like to extend this work to larger scale network models such as network of queues. Lastly we plan to evaluate FMDP online on very large scale cloud scenarios with deep Reinforcement Learning methods.

CHAPTER 7

CAUSAL REINFORCEMENT LEARNING FOR MULTI-TIER NETWORK

Up to now, we have seen different model-based approaches to learn auto-scaling policies in Cloud environments such as a one physical node system or a more complex scenario with multi-tier architectures. The major issues encountered is about the *curse of dimensionality* that occurs when we consider larger scale Cloud environments. First model-free RL methods highly suffer to converge. Secondly, the use of tabular model-based RL algorithms (see Chapter V) can improve performance regarding speed of convergence and policy quality but still have trouble to handle very large scale scenarios, precisely when addressing learning objects implementation but also convergence. Instead of investigating the approximation path for improvement with neural networks as a representative tool of environment models, we decided to focus on the structural properties of the environments to be integrated in the learning agent's knowledge. We considered the factored framework developed in Chapter VI which has obtained good performance regarding the improvement for implementation and considering higher scale scenarios. Yet the factored representation is still a tabular approach and also has difficulty in addressing very large problems. We believe that a causal approach can provide even better enhancements by going further in the understanding of environment variables relations. This causal dimension goes beyond the associational representation proposed in the factored framework. The benefits can be numerous. Among those are the increased capability of copying with high-scale systems, the explainability of the obtained policies, improved convergence speed, as well as capability of dealing with hidden variables or unobserved confounders. Our focus in this chapter will be the two last aspects. In this regard, this chapter acts as an orthogonal view : we are more concerned with MDP environments with particular structure (hidden variables, unobserved confounders, etc.) than very large systems.

Therefore in this chapter we propose to unify causal reasoning in reinforcement learning algorithms to accelerate convergence with counterfactuals and deal with Cloud environments with unobserved state variables that can have confounding effects or not. We first provide more detailed background about the *Causality* domain then propose a generalised formalism for MDP environment representation with *Structural Causal Models* (SCM) that are defined in the causality background section. After this combination of causality and reinforcement learning formalism, we evaluate *Causal Reinforcement Learning* (CRL) algorithms on the multi-tier Cloud environment, considering several scenarios : the initial tandem queue, the partially observable tandem queue with MMPP arrivals and the tandem queue with unobserved confounders.

The chapter contributions are listed in the following :

* Generalisation of Structural Causal Model formalism for any MDP environment and Counterfactual reasoning;

* Proposal of a counterfactual model-free RL algorithm;
* Assessment of causal RL methods in tandem queue environments.

7.1 Introduction to causality : Pearl's Causal Hierarchy

In order to fully understand the integration of causal reasoning in the reinforcement learning scheme, we first provide elements of causality. Cause-and-effect relationships play a key role in how we perceive and make sense of the world around us, how we interact with it and, ultimately, how we comprehend ourselves. Almost two decades ago, computer scientist Judea Pearl [3] made a breakthrough in understanding causality by proposing and studying the "Ladder of Causation", a framework that showcases the distinct roles of seeing, doing, and imagining. We summarise in this section his vision and its proposal for a mathematical formalism of causality, namely structural causal models. For this purpose, we follow the description given by Bareinboim et al. in [20]. A SCM naturally defines a hierarchy of concepts, described as the "ladder of causation" which Bareinboim et al. have been calling the *Pearl Causal Hierarchy*, or PCH. There exist three layers : *Associational*, *Interventional* and *Counterfactual* (see Figure 7.1).

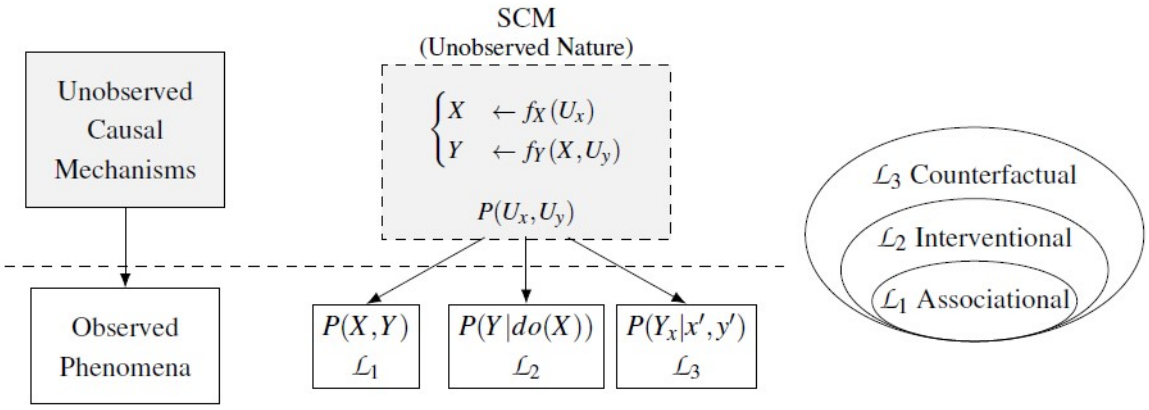


FIGURE 7.1 – Pearl's causal hierarchy representation of the world (Figure from [20]

Definition 12 (Structural Causal Model [20]). We define a structural causal model SCM $\mathcal{M}_C$ as a 4-tuple $< V, U, \mathcal{F}, P(U) >$ where :

— $U = \{U_1, U_2, \ldots, U_k\}$ is a set of exogenous (unobserved) variables, determined by factors outside the model;

— $V = \{V_1, V_2, \ldots, V_n\}$ are called endogenous (observed) and are determined by other variables in the model – that is, variables in $U \cup V$;

— $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is a set of causal assignments such that f_i maps variables from $U_i \cup Pa_i$ to V_i. $\forall i = 1, \ldots, n, v_i \leftarrow f_i(Pa_i, u_i)$, i.e. the value of v_i will change caused by its parents Pa_i and noise u_i;

— $P(U)$ is the distribution of noises variables.

It is assumed that the causal assignments f_i are invariant, except if intervention forces them to change.

7.1.1 Layer 1 - Observational

If you let the system behaves by itself and just observe it evolving, then you can observe the joint distribution of endogenous variables $P(V)$ which is defined by :

Definition 13 (Observational distribution [20]). An SCM $\mathcal{M}_C =< U, V, \mathcal{F}, P(U) >$ defines a joint probability distribution $P^{\mathcal{M}_C}(V)$ such that for each $Y \subseteq V : P^{\mathcal{M}_C}(y) = \sum_{u|Y(u)=y} P(u)$ where $Y(u)$ is the expression for Y after evaluation with $\mathcal{F}$ and $U = u$.

In this setting, we only rely on *observational* data with no interventions in the system.

7.1.2 Layer 2 - Interventional

Secondly, one agent (human, software) can interact with an environment by intervening in the system. The data collected in this setting will be *interventional*.

Definition 14 (Interventional SCM [20]). Let $\mathcal{M}$ be a causal model, X a set of variables in V, and x a particular realisation of X. A submodel $\mathcal{M}_x$ of $\mathcal{M}$ is the causal model $\mathcal{M}_x =< U, V, \mathcal{F}_x, P(U) >$, where $\mathcal{F}_x = \{f_i : V_i \notin X\} \cup \{X \leftarrow x\}$. In other words, an interventional SCM has a subset of endogenous variables where their causal assignments has been changed by intervention.

Notice that this intervention is a *do-operator* which in Pearl's terminology is a mathematical operator, denoted $do(X = x)$ which simulates physical interventions by deleting certain functions from the model, replacing them with a constant $X = x$, while keeping the rest of the model unchanged. Naturally, this interventional world is similar to reinforcement learning/MDP scenarios where an autonomous agent interacts with the environment by intervening in the system, therefore changing directly values of some endogenous variables or forcing causal assignments to change. However, PCH goes further and define a third layer : the counterfactual world that gives the ability to learning agents for imagining.

7.1.3 Layer 3 - Counterfactual : the imaginary world

The counterfactual mode is an imaginary world where an agent can measure the possible effect of non-taken interventions, that is, thinking about alternative ways the world could be. In other worlds, it can evaluate the outcomes of imaginary decisions that would have happened in a given situation (or *evidence*).

Definition 15 (Evidence). We define $\mathcal{E}$ an evidence of a system $\mathcal{M}$ the realisations of the environment variables at a given time t. The evidence $\mathcal{E}$ is therefore the expression of the endogenous and exogenous variables $\mathcal{E} = \{u_1, \ldots, u_k, v_1, \ldots, v_n\}$. It does not depend on whether there was an intervention or not in the system; it only depicts the systems values at a given time.

In the RL paradigm, when the learning agent interacts with the environment, it observes the state of the system s and decides to act according to its policy $q(s)$. Rollouts (or trajectories) resulting from this state-action pair can be propagated backwards for updating value functions or policies (as we have seen thorough this document with model-free or model-based RL techniques). However, this backpropagation update is done based on the fact that there was a specific evidence $\mathcal{E}$ at time t when agent did action a in state s. One possibility to update the value function for this pair (s, a) is to visit it infinitely many times (which is an objective in RL for convergence). However, counterfactual reasoning can do this update in the meantime by evaluating the outcome (new state and reward) of other actions not-taken and under the same evidence that occurred. This reasoning brings a supplementary knowledge and more powerful learning process since the agent can evaluate the outcomes of many decisions in a given state s and under the same evidence. The counterfactual reasoning is possible if the agent is provided the SCM of the world and is based on three principles that we describe below :

— **Abduction** : Discover noise realisations u after deciding action a in the environment v_a. In other words, the agent discovers the evidence $\mathcal{E}$ of the system and more generally updates conditional probability distributions $p(u|v_a)$;

— **Intervention** : Intervene in the environment by changing f_i for some endogenous variables v_i. This is usually a new decision a' ;

— **Evaluation-Prediction** : Predict or evaluate the outcome $v_{a'}$ after intervention in the same context $p(u|v_a)$.

We provide a simple example to illustrate the counterfactual reasoning.

Example 4. *Imagine a patient named Joe has taken a treatment* $(X = 1)$ *and died* $(Y = 1)$. *The question counterfactual reasoning asks is the following : What is the probability that Joe would have survived* $(Y = 0)$ *had he not taken the treatment* $(X = 0)$, *denoted* $\mathcal{P}(Y_{X=0} = 0|X = 1, Y = 1)$ *?*

Assume the following SCM model :

$$\mathcal{M} = \begin{cases} x = u_1 \\ y = xu_2 + (1 - x)(1 - u_2) \end{cases}$$

The first counterfactual step abduction is used to update the probability $\mathcal{P}(u)$ *knowing the evidence of the system* e. *In this example we have the following observation of the system* $e = \{x = 1, y = 1\}$. *From the SCM, we can retrieve the noise values and infer that* $\{u_1 = 1, u_2 = 1\}$.

The second step intervention would be to test the action that has not been taken, namely $(X = 0)$ *here. Therefore, we can replace* x *in the SCM as follows :*

$$\mathcal{M} = \begin{cases} x = 0 \\ y = xu_2 + (1 - x)(1 - u_2) \end{cases}$$

Finally, we can do the prediction phase by evaluating the outcome Y *from the noise discovery* $u = \{u_1 = 1, u_2 = 1\}$ *and the new intervention* $(X = 0)$ *:* $\mathcal{M} = \begin{cases} x = 0 \\ y = x * 1 + (1 - 0)(1 - 1) \end{cases}$

which leads to :

$$\mathcal{M} = \begin{cases} x = 0 \\ y = 0 \end{cases}$$

The conclusion of the counterfactual reasoning is that Joe would have been alive if he had not taken the treatment.

Finally, we display in Figure 7.2 a summary of the three causal layers and their association with optimisation techniques.

	Layer (Symbolic)	Typical Activity	Typical Question	Example	Machine Learning
$\mathcal{L}_1$	Associational $P(y\|x)$	Seeing	What is? How would seeing X change my belief in Y?	What does a symptom tell us about the disease?	Supervised / Unsupervised Learning
$\mathcal{L}_2$	Interventional $P(y\|do(x), c)$	Doing	What if? What if I do X?	What if I take aspirin, will my headache be cured?	Reinforcement Learning
$\mathcal{L}_3$	Counterfactual $P(y_x\|x', y')$	Imagining	Why? What if I had acted differently?	Was it the aspirin that stopped my headache?	Causal Reinforcement Learning

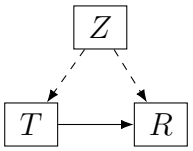
FIGURE 7.2 – Summary of the three causal layers [20]

7.1.4 Why reinforcement learning can suffer for policy optimisation

We provide in this section a well-known example in the Causality domain that explains the effects of confounding variables and their impact on the policy to be learned. This example is given in order to motivate the integration of causality in the RL field.

7.1.4.1 The kidney stone example

We illustrate in this section why reinforcement learning agents can suffer to learn efficient policies in specific environments and why a causal knowledge can help the learning agent to improve its policy. The *kidney stone example* illustrates the *Simpson's paradox*[1]. A doctor can administrate to a patient either treatment A or treatment B. A confounder variable Z representing the size of the kidney stone can influence the physician's decision T ($T = A$ or $T = B$) and the recovery R (0 or 1).



After experiments, we observe in Figure 7.3 that patients have higher recovery rate with treatment A when we look separately for different stones size (small, large). But if we look at the whole population, patients have higher recovery rate with treatment B. Therefore there is a paradox (Simpson's paradox) on what treatment the physician should administrate.

Formally, we have :

$$p(R = 1|T = b) > p(R = 1|T = a)$$

but

$$p(R = 1|T = b, Z = l) < p(R = 1|T = a, Z = l) \text{ and } p(R = 1|T = b, Z = s) < p(R = 1|T = a, Z = s).$$

	Treatment A	Treatment B
Small stones	93% (81/87)	87% (234/270)
Large stones	73% (192/263)	69% (55/80)
Combined	78% (273/350)	83% (289/350)

FIGURE 7.3 – Recovery rate for patients in kidney stone example

One intuitive interpretation of this kidney stone example of Simpson's paradox is that large stones are more severe than small stones and are much more likely to be treated with treatment A, making treatment A seem worse overall than treatment B. A second possible explanation is that treatment B is less expensive and less intrusive (e.g., an oral pill), so it is more commonly used for less severe cases, including most cases of small kidney stones. Doctors therefore prefer to avoid treatment A, which is very efficient but much more costly and invasive (e.g., surgery), except in the most difficult cases.

Note that the data gathered in this example comes from physician observation in a hospital, which means that treatments are assigned based on physician thinking and current norms of care. Physicians allocate treatments in part based on the severity of the patient's case, but the severity of the patient's case also influences the patient's

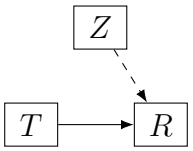
1. https ://plato.stanford.edu/entries/paradox-simpson/

chances of recovery. In contrast, a randomized trial, in which we randomly assign treatments to patients regardless of the size of the kidney stone, will break down any associations that may exist between treatment assignment and severity; therefore, it will undermine the confounding effect. However, this random exploration is avoided in most real-world environments for obvious reasons such as safety to cite one example (healthcare, autonomous vehicles, etc.). Thus, in many use cases, solutions must be found to deal with the confounding effect.

7.1.4.2 How to solve Simpson's paradox : Counterfactual or Adjustment Set

We can overcome this issue by evaluating the distribution of outcome $P(R|T)$ by integrating confounder. In the kidney stone example, we evaluated $p(R = 1|T = b, Z = l)$ and $p(R = 1|T = a, Z = l)$ and compared the conclusion with evaluation of $P(R = 1|T = a)$ and $P(R = 1|T = b)$. We observe that the conclusions are different from both approaches : Treatment A is better when considering kidney stones values while Treatment B is better if we omit these values.

Now to evaluate the effects of treatment on the recovery rate $P(R|do(T = t))$, one can use *adjustment set* technique where confounder is observable.



By intervening on variable T, we break influence of Z on T, rending Z and T completely independent. Therefore we can evaluate :

$$P(R|do(T)) = \sum_z p(R|do(T), Z)P(do(T), Z) = \sum_z p(R|do(T), Z)P(do(T))P(Z) = \sum_z p(R|do(T), Z)P(Z)$$

which differs from :

$$P(R|T) = \sum_z p(R|T, Z)P(T, Z) = \sum_z p(R|T, Z)P(T)P(Z|T) = \sum_z p(R|T, Z)P(Z|T)$$

Evaluating $P(R|do(T))$ with *adjusment set* technique will give the right conclusion. Now, this technique assumes first that confounder can be observable and are used in observations settings which enables the estimation of $P(R|do(T))$ without intervention. However in many cases these values are unobserved and this technique can't be applied. Thus counterfactual reasoning can be applied if causal model is known. We describe here an idea of couterfactual reasoning for this example yet the framework will be further developed in Section 7.2.5. One assume that an agent knows causal relation $R = f_R(T, Z, U)$ where U is a noise expressing pure randomness of the system. Given a dataset with observation tuples $\{t, r\}_i$, one can retrieve realisations of z by computing $z = f_R^{-1}(r, t)$. By doing so, we will retrieve a value for (z, u) which will be z if the system is purely deterministic with $U = 0$ w.p. 1. If we have computed realisations of Z values for data $\mathcal{D}_i$, we can infer conditional distribution $P(Z|\hat{x}_0)$ where $\hat{x}_0$ are observations (t, r). Then agent can do counterfactual planning by evaluating do-operations $(do(T = A), do(T = B)$ e.g.) in SCM system with $f_R, P(Z|\hat{x}_0)$. Moreover, in RL we won't need this adjustment formula as we will estimate $P = (R|do(T))$ directly through interventions.

To summarise, hidden variables with confounding effects may be one of the reasons why reinforcement learning severely suffers in real industrial use-cases and having a causal model of the world for counterfactual reasoning could be one major improvement to encourage the utilisation of RL algorithms in practice.

7.2 Causal Reinforcement Learning : Formalism and Environments

This section ought to build a general formalism for Reinforcement Learning methods with causal model approaches. We first give an overview of the recent literature works about Causal Reinforcement Learning. Then, we display the main difference for representing Markov Decision Process environment with usual models and with causal models. Afterwards, we describe possible existing MDP environments, next provide definitions for SCM equivalent description of the MDP environment models. Finally, we provide the counterfactual reasoning process in different MDP environments.

7.2.1 Causal Reinforcement Learning literature

Very recently, researchers [30, 47, 153] ought to merge the causal and reinforcement learning fields. To introduce this new emerging field, we first provide understanding about causal and initial MDP representations. Secondly, we give a well-known example in the causality field that can explain why RL can fail in some environments. Next, we give an overview of very recent works that studied this integration by providing CRL techniques and formalism. We have seen that reinforcement learning domain was associated with interventional layer 2 of the causality domain. As for the kidney stone example where counterfactual reasoning can bring benefit for the optimal policy search, the idea of Causal Reinforcement Learning is to bring this counterfactual reasoning in reinforcement learning methods. This emerging domain has received very few attention in the past years but starts to grow. The works [153] and [30] are two works dealing with causal reinforcement learning and counterfactual reasoning. They both demonstrates that counterfactual reasoning can bring policy optimisation improvement in specific scenarios, compared to SoTA reinforcement learning methods. Bareinboim et al. [153] focus on Markov decision processes with unobserved confounders ($\mathcal{MDPUC}$) environment, where a set of endogenous variables $\mathcal{H}$ is hidden to the RL agent and these variables confound the agent's policy $q(\mathcal{S}, \mathcal{H})$ and the reward $\mathcal{R}$. However, in their formalism they integrate unobserved confounders (UC) inside exogenous set of variables, regarding the SCM definition 12. On the other hand, the Deepmind team [30] restricts its scope on partially observable Markov decision process ($POMDP$) environment where they integrate a noise hiding some state features to the agent in the SOKOBAN environment [2].

Both papers represents the MDP environment with structural causal model (SCM), described in section 7.1.

Definition 16 ([153]). A Markov Decision Process with Unobserved Confounders (MDPUCs) is an augmented SCM $\mathcal{M}$ with finite action domain $\mathcal{A}$, state domain $\mathcal{S}$, and reward variable R :

- $\gamma \in [0, 1)$ is the discount factor;
- $U(t)$ is the exogenous variable at round t;
- $V(t) = \mathcal{A}(t) \cup R(t) \cup \mathcal{S}(t)$ is the set of endogenous (observable) variables at round t with $\mathcal{A}(t)$ the action space, $\mathcal{S}(t)$ the state space and $R(t)$ the reward function;
- $F = \{f_a; f_y; f_s\}$ is the set of structural equations relative to V such that $A(t) = f_a(s(t); u(t)) = q(s(t); u(t)), R(t) = f_r(a(t); s(t); u(t))$ and $S(t) = f_s(a(t-1); s(t-1); u(t-1))$;
- P(u) encodes the probability distribution over the exogenous variables U.

The work [30] also consider an SCM modelling of the MDP environment but does not formally define this translation $MDP \rightarrow SCM$. Yet they propose in their paper a solution to equivalently describe state-transitional models $\mathcal{M}$ with causal assignments. For generalisation of dynamics modelling in causal model and state-transitional model, one can represent both structure equivalently. In [30] is stated a lemma for this equivalent representation.

2. https ://fr.wikipedia.org/wiki/Sokoban

Lemma 5 (Auto-regressive uniformisation [30]). *Consider random variables $X_1, \dots, X_N$ with joint distribution P. There exist functions f_n for $n \in \{1, \dots, N\}$ such that with independent random variables $U_n \sim$ Uniform[0,1], the random variables X' equal X in distribution, i.e. $X' \overset{d}{=} X$, where $X' = (X'_1, \dots, X'_n)$ are defined as : $X'_n := f_n(U_n, X'_{<n})$.*

In other words, we can express relations between state variables by a functional form, whereas in state-transitional model we only express the possible future value of a variable at next time step with probabilities distribution. Taking functional assignment between state variables allows to express time causal relation and to decompose causal link and noise. The reverse is always true since we can derive probability distribution from the functional links. Moreover, this ensure that the two dynamics representation are equivalent, meaning that both formalism express the same environment dynamics. However, they represent the same environment at different level of knowledge, since learning agents can perform counterfactual in the new causal representation.

7.2.2 Markov decision process environment's representation

As described in the Definition 12 of Structural Causal Models, the dynamics and reward functions are depicted in the system with causal assignments f_i and exogenous and endogenous variables U and V. This is a fundamentally different representation for the MDP environments. This approach separates causal determinism and system's randomness whereas it is represented as single objects in classical reinforcement learning models. To fully distinguish the two approaches, we redefine in Definition 17 the usual MDP models, denoted now *State-Transitional* models.

Definition 17 (State-transitional model). We define the state-transitional model $\mathcal{M}$ of the environment to be the initial representation of the MDP environment, namely the transition and reward functions : $\mathcal{M} = \{\mathcal{P}, \mathcal{R}\}$.

In usual MDP environments, a state-transitional model acts as a predictor that can give the new state s' and reward r when agent acts action a in state s : $(s', r) \leftarrow \mathcal{M}(s, a) \iff (s', r) \leftarrow \mathcal{P}(s, a), \mathcal{R}(s, a)$. To be consistent with the works described in this document, we assume also that the reward function is stationary and that $p(\mathcal{R}(s, a) = r) = 1$ for all state-action pairs. This is why we highlight only the difference in the dynamics representation of the system. The state-transitional dynamics $\mathcal{P}(s'|s, a)$ represented by a conditional distribution represents the probability to move in state s' knowing that the agent has done action a in state s. This object integrates both the environment determinism and randomness and the two aspects can not be differentiated.

On the contrary, the causal model $s' = f_s(s, a, u)$ represented by a causal assignment, provides the future value of state s' knowing that the agent has done action a in state s. The causal model can differentiate the deterministic part from the intrinsic randomness of the system, represented by exogenous noise u. This representation difference is illustrated in Figure 7.4.

7.2.3 MDP Environments

To represent the variability of environments available in nature, we define three possible environment scenarios. The first application is a system *fully observable* by the agent (MDP), i.e. the agent can observe all system's variables including : state variables, reward variables. The second application *partially observable* systems (POMDP) is therefore similar but some endogenous variables are hidden to the agent at any epoch. The last application is MDP with *unobserved confounders* (MDPUC) where some endogenous environment variables are never observed (e.g. difficult to measure) but yet can influence outcome, other state variables, or even an expert's decision policy (expert can imagine possible value of a variable), thus confounding the effect of a decision on the environment, rendering the evaluation biased. In other words, in POMDP, hidden variables does not have a confounding effect while they do in MDPUC scenarios.

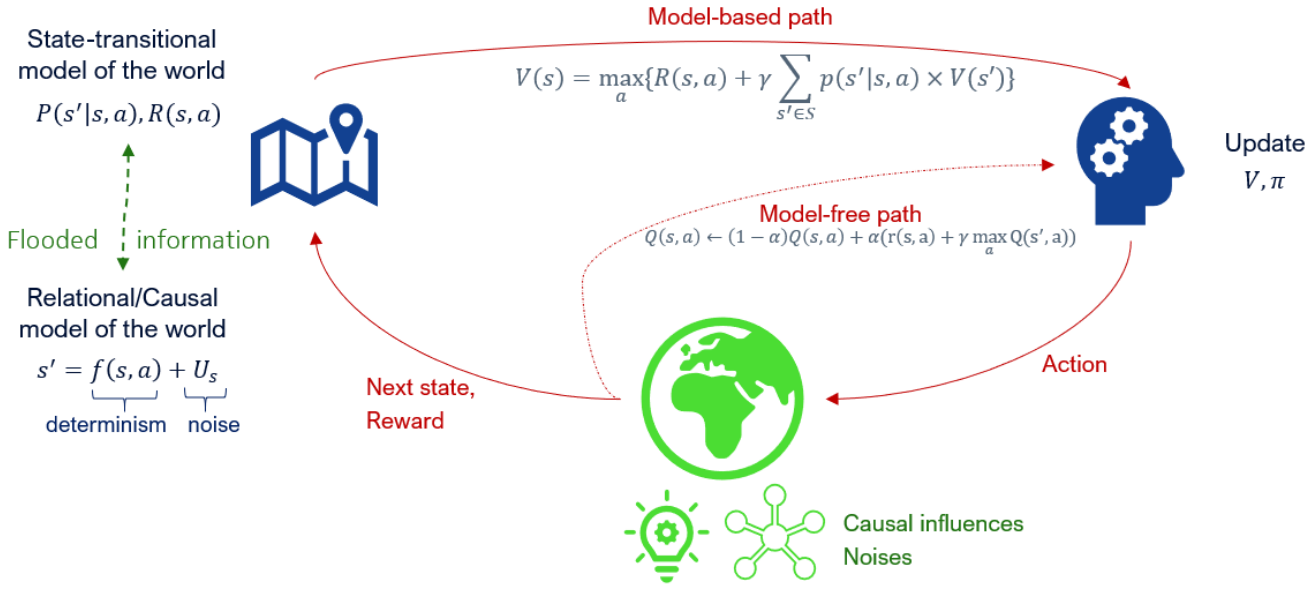


FIGURE 7.4 – Two different representations of the environment dynamics

In addition, we denote the intrinsic difference between contexts by a change of the letter, and we denote a change in the state space type by a number 1 or 2, where 1 deals for countable discrete state space and 2 deals for uncountable discrete or continuous state space. We cautiously differentiate state space types for agent's representation of the model, value function and policy. We call exogenous variables expressing the randomness of the systems by $\mathcal{U}$, endogenous variables expressing all environment variables by $\mathcal{V}$.

Definition 18 (MDP Environments). All MDP environments in nature can be described by :

- **MDP**. $\mathcal{E}nv\ \mathcal{A}$. Environments with fully observable $\mathcal{V}$:
  They are fully observable, i.e. the RL agent can observe all variables that exist in the environment and its policy is based on the full observation of the system. We denote by $\mathcal{A}_1$ environments with countable discrete state space and by $\mathcal{A}_2$ environments with continuous/uncountable discrete state space.
- **POMDP**. $\mathcal{E}nv\ \mathcal{B}$. Environments with partially observable $\mathcal{V}$ and without unobserved confounders :
  They are partially observable, i.e. the RL agent can only observe at a given time t a subset of variables that exist in the environment and its policy is based on the partial observation of the system. In this setting, the Markovian property is violated (more explanations in Section 7.2.4). Moreover, no hidden variables have confounding properties. We denote by $\mathcal{B}_1$ environments with countable discrete state space and by $\mathcal{B}_2$ environments with continuous/uncountable discrete state space.
- **MDPUC**. $\mathcal{E}nv\ \mathcal{C}$. Environments with partially observable $\mathcal{V}$ and with unobserved confounders :
  They are partially observable, i.e. the RL agent can only observe at a given time t a subset of variables that exist in the environment and its policy is based on the partial observation of the system. In this setting, the Markovian property is still valid (more explanations in Section 7.2.4). Moreover, at least one hidden variable is a confounder that can confound several endogenous variables $v \in \mathcal{V}$ (state variables, reward, policy). We denote by $\mathcal{C}_1$ environments with countable discrete state space and by $\mathcal{C}_2$ environments with continuous/uncountable discrete state space.

We will detail the difference between POMDP and MDPUC in the structural causal model representation displayed in the next section 7.2.4 but still give a little explanation here. As we have seen, there seems to be two kind of considered use-cases when researchers tackle causality in reinforcement learning : POMDP [30, 154] and

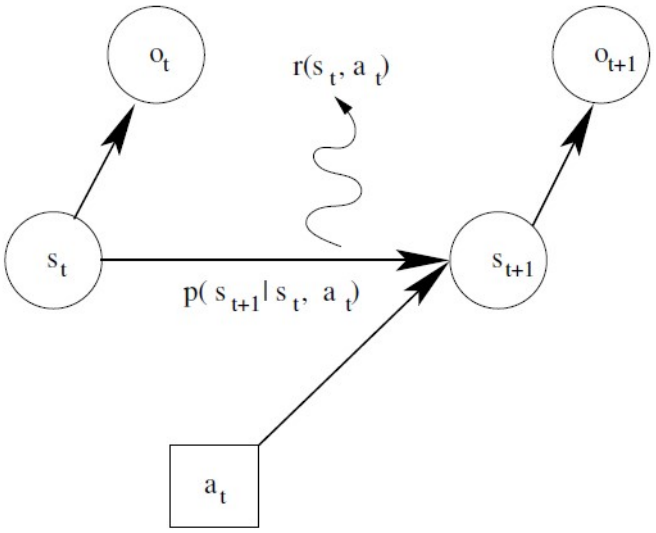


FIGURE 7.5 – Partially observable MDP transition from t to $t+1$ (Figure from [2])

MDPUC [97, 153]. We propose to differentiate POMDP and MDPUC by the nature of the hidden variables, i.e. that they should have properties such that in POMDP scenario the Markovian property is violated while it is maintained in the MDPUC scenario.

A POMDP [2] is a MDP in which the agent does not observe the real environment state. The general principle, illustrated in Figure 7.5, is that at each time t, the agent does not know the current state $s^t \in S$ but can only partially perceive it with an observation denoted o^t. This observation is given by the observation function $O(s^t)$. When the learning agent applies an action $a^t \in \mathcal{A}$ on the process, it then randomly modifies the the state of the process according to the environment transitions $\mathcal{P}()$ to bring it in the state s^{t+1} while the agent only perceives the observation o^{t+1}. Finally, the agent receives a reward r as in the MDP setting. Moreover, even if the observation function $O()$ is deterministic i.e. each state is associated with one and only one observation, the agent may not know the state. Therefore, two states can be associated to the same observation which generates ambiguity and can lead to poor quality of the policy. This can be simply understand by looking at the policy learned from the agent's perspective. It updates a policy $q : O \rightarrow \mathcal{A}$ such that for each observation o it generates a decision a. Now imagine that for a given observation o_1 is associated two states s_{11} and s_{12}. This means the agent has learned after the learning process to do $q(o_1)$ which may be optimal in the state s_{11} but very bad in s_{12}. This is why in POMDP settings, algorithms [2] often deal with history of observations such that the agent can have a clue in which state he is regarding the trajectory. Intuitively, this is a solution to overcome the violated Markovian property.

Notice also that Bareinboim et al. [153] already provided first explanations about the differences between POMDP and MDPUC environments. According to the authors, POMDP are MDPs with partial or no information of the state variables, where the partial observation does not summarise all the trajectories that led to the present state (Markovian property is violated). MDPUC are MDPs with local unobserved variables that confound the relationships between actions $a(t)$, effects $\mathcal{R}^{t+1}$, and states s^{t+1}. They claim two key differences between POMDP and MDPUC. First, POMDP do not necessarily imply the confounding, i.e., it is possible that no knowledge about the state variable is available, but there exists still no unobserved confounding in the system. Secondly, the authors demonstrate in [153] that the Markovian property holds in MDPUC while it is clearly violated in POMDP.

7.2.4 SCM representation of MDP environments

We build a bridge between the three families of MDP environments : MDP, POMDP and MDPUC by formalising a general SCM for any MDP environment models. Formally, we differentiate unobserved confounders from noise exogenous variables and consider them as endogenous variables that can be unobserved by the agent. Thus we consider a physical system or environment $\mathcal{E}$ by having two distinct parts. The first one is the deterministic causal relation part with causal assignments F between environment endogenous state variables $S = (s_1, ..., s_n)$, reward variable R which is a function of state variables s_i and action variables $A = (a_1, ..., a_l)$. The second class represents natural inherent randomness of the system with exogenous noise variables $U = (u_1, ...u_k)$ that can disturb deterministic relations.

Definition 19 (Markov Decision Process with SCM). A Markov Decision Process can be represented by a structural causal model $\mathcal{M}$ with finite action domain $\mathcal{A}$, state domain $\mathcal{S}$, and reward variable R :

— $\gamma \in [0, 1)$ is the discount factor;

— U^t is the set of exogenous variables at time t determined by factors outside the model and representing the randomness of the system. All noise variables u are independent from each other;

— $\mathcal{V}^t = \mathcal{A}^t \cup R^t \cup \mathcal{S}^t$ is the set of endogenous (potentially observable) variables at time t. This includes all environment variables potentially observable by the agent;

— $F = \{f_a; f_r; f_s\}$ is the set of structural equations relative to V such that $a^t = f_a(s^t; u_a^t) \sim q(a^t|s^t)$, $R^t = f_r(a^t; s^t; u^t) \sim P(R^t|s^t, a^t)$ and $S^{t+1} = f_s(a^t; s^t; u^t) \sim P(s^{t+1}|s^t, a^t)$. Note that policy q is deterministic if $u_a^t = 0$ else stochastic;

— P(U) encodes the probability distribution over the exogenous variables U.

Definition 19 defines the structural causal model representation of a MDP where the state s is represented as a single object. We have seen in Chapter VI that the state could be represented by a vector of state variables s_i in the factored approach. For this purpose, we also propose a definition for structural causal model representation of factored MDP. And finally, this latest formalism is the most suitable since we will also deal with causal graphs and local causal relations in the SCM approach. Therefore, the following definition 20 will be the main formalism for MDP representation by SCM.

Definition 20 (Causal Markov Decision Process). A Factored Markov Decision Process can be represented by a structural causal model $\mathcal{M}$ with finite action domain $\mathcal{A}$, state domain $\mathcal{S} = \{s_1, \ldots, s_N\}$, and reward variable R :

— $\gamma \in [0, 1)$ is the discount factor;

— $U^t = \{u_1^t, \ldots, u_k^t, u_a^t, u_r^t\}$ is the set of exogenous variables at time t determined by factors outside the model and representing the randomness of the system. All noise variables u are independent from each other;

— $\mathcal{V}^t = \mathcal{A}^t \cup R^t \cup \mathcal{S}^t$ is the set of endogenous (potentially observable) variables at time t. This includes all environment variables potentially observable by the agent. If some variables in $\mathcal{V}$ are hidden at some times t, we are in POMDP or MDPUC;

— $F = \{f_a; f_r; f_{s_1}, \ldots, f_{s_N}\}$ is the set of structural equations relative to $\mathcal{V}$ such that $a^t = f_a(s^t; u_a^t) \sim q(s^t)$, $\mathcal{R}^{t+1} = f_r(a^t, Pa(\mathcal{R}^t); u_r^t) \sim P(R^{t+1}|Pa(R^t), a^t)$ and
$\forall i = \{1, \ldots, N\}, s_i^{t+1} = f_{s_i}(a^t; Pa(s_i^t); u_i^t) \sim P_i(s_i^{t+1}|Pa(s_i^t), a^t)$
, where $Pa(.) \in \mathcal{S}$ denotes the parents variables values at previous time step;

— P(U) encodes the probability distribution over the exogenous variables U.

In order to treat specific environments, we provide more explanations to make a clear distinction between partially observable environments with no confounders and with confounders. Indeed, POMDP and MDPUC are specific environments that can also be described by definitions 19, 20 with supplementary constraints or hypothesis on the environment endogenous variables $\mathcal{V}$. We provide definitions for the two components with the factored framework and where there is only one hidden variable. Recall that in the POMDP scenario, the Markovian property is violated. On the contrary, in MDPUC, the Markovian property is still valid, and missing information about the hidden variable does not avoid to fully grasp what will happen in the next time step $t+1$.

Definition 21 (Causal Partially Observable Markov Decision Process). A Factored Markov Decision Process can be represented by a structural causal model $\mathcal{M}$ with finite action domain $\mathcal{A}$, state domain $\mathcal{S} = \{s_1, \ldots, s_N\}$, and reward variable R :

— $\gamma \in [0, 1)$ is the discount factor;

— $U^t = \{u_1^t, \ldots, u_k^t, u_a^t, u_r^t\}$ is the set of exogenous variables at time t determined by factors outside the model and representing the randomness of the system. All noise variables u are independent from each other;

— $\mathcal{V}^t = \mathcal{A}^t \cup R^t \cup \mathcal{S}^t$ is the set of endogenous (potentially observable) variables at time t;

— $z^t \in \mathcal{V}^t$ a hidden variable. This variable can be derived from external factors outside the environment but has to be influenced by itself and eventually by other endogenous variables, i.e. $z^{t+1} = f_z(z^t, Pa(z^t), u_z^t)$ and $\exists v_i \in V$ s.t. $z^t \in Pa(v_i^{t+1})$. This ensures that the Markovian property is violated;

— $F = \{f_a; f_r; f_{s_1}, \ldots, f_{s_N}\}$ is the set of structural equations relative to V such that $a^t = f_a(s^t; u_a^t) \sim q(s^t)$, $R^{t+1} = f_r(a^t, Pa(R^t); u_r^t) \sim P(R^{t+1}|Pa(R^t), a^t)$ and
$\forall i = \{1, \ldots, N\}, s_i^{t+1} = f_{s_i}(a^t; Pa(s_i^t); u_i^t) \sim P_i(s_i^{t+1}|Pa(s_i^t), a^t)$;

— P(U) encodes the probability distribution over the exogenous variables U.

In the POMDP scenario, missing the value of z will break the Markovian property of the system since missing this information will collapse the knowledge about state transitions. Therefore the learning agent will have to treat the problem with history of observations to have information about dynamics and missing value. This is done in the work proposed by Deepmind [30]. Now, the difference with unobserved confounder scenario is that the Markovian property is not broken. The UC variable follows a distribution with external factors outside the environment. In other words, missing the value of the UC variable does not introduce a lack of information for the state transition from step t to $t+1$ as it does in the POMDP scenario.

Definition 22 (Causal Markov Decision Process with Unobserved Confounder). A Factored Markov Decision Process can be represented by a structural causal model $\mathcal{M}$ with finite action domain $\mathcal{A}$, state domain $\mathcal{S} = \{s_1, \ldots, s_N\}$, and reward variable R :

— $\gamma \in [0, 1)$ is the discount factor;

— $U^t = \{u_1^t, \ldots, u_k^t, u_a^t, u_r^t\}$ is the set of exogenous variables at time t determined by factors outside the model and representing the randomness of the system. All noise variables u are independent from each other;

— $\mathcal{V}^t = \mathcal{A}^t \cup R^t \cup \mathcal{S}^t$ is the set of endogenous (potentially observable) variables at time t;

— $(UC)^t \in \mathcal{V}^t$ an unobserved confounder that influences other environment variables in V^t. This variable should be derived from external factors outside the environment, i.e. following its own distribution, i.e. $(UC)^t \sim \mathbb{P}_{UC}$. This ensures the Markovian property to be valid.

— $F = \{f_a; f_r; f_{s_1}, \ldots, f_{s_N}\}$ is the set of structural equations relative to V such that $a^t = f_a(s^t; u_a^t) \sim q(s^t)$, $R^{t+1} = f_r(a^t, Pa(R^t); u_r^t) \sim P(R^{t+1}|Pa(R^t), a^t)$ and
$\forall i = \{1, \ldots, N\}, s_i^{t+1} = f_{s_i}(a^t; Pa(s_i^t); u_i^t) \sim P_i(s_i^{t+1}|Pa(s_i^t), a^t)$;

— P(U) encodes the probability distribution over the exogenous variables U.

We have so far proposed SCM formalism for three types of MDP environments : MDP, POMDP and MDP-PUC. We want to see now how the learning agent can do counterfactual reasoning to ameliorate RL algorithms performance in each of these environments.

7.2.5 Counterfactual reasoning in MDP environments

We display in this section the counterfactual reasoning framework in all scenarios MDP, POMDP and MDP-PUC. We first provide a toy example to illustrate the counterfactual process then display RL methods to integrate the counterfactual reasoning in the different MDP environments.

7.2.5.1 Toy example

We provide a toy example on these three scenarios for better understanding. We consider an environment with two state variables $X = (X_1, X_2)$, an action variable A and an outcome/reward variable R. Formally $\mathcal{V} = X \cup A \cup R$. Next we consider exogenous noise variables U with u_1, u_2 respectively for expressing the randomness for state variables X_1 and X_2, u_a for expressing the stochasticity of agent's policy q and u_r the randomness for the outcome variable. We draw the causal graph for the initial MDP case in Figure 7.6 and provide the associated structural causal model with $P(U)$ unknown :

$$\mathcal{M}_{MDP} = \begin{cases} A^t = q(X_1^t, X_2^t, u_a^t) \\ X_1^{t+1} = f_1(X_1^t, X_2^t, A^t, u_1^{t+1}) \\ X_2^{t+1} = f_2(X_2^t, A^t, u_2^{t+1}) \\ \mathcal{R}^{t+1} = f_r(A^t, X_1^t, X_2^t, u_r^{t+1}) \end{cases}$$

Next, we display CRL algorithms that handle this counterfactual process for each environments and that will be applied on the tandem queue scenario in Section 7.4.

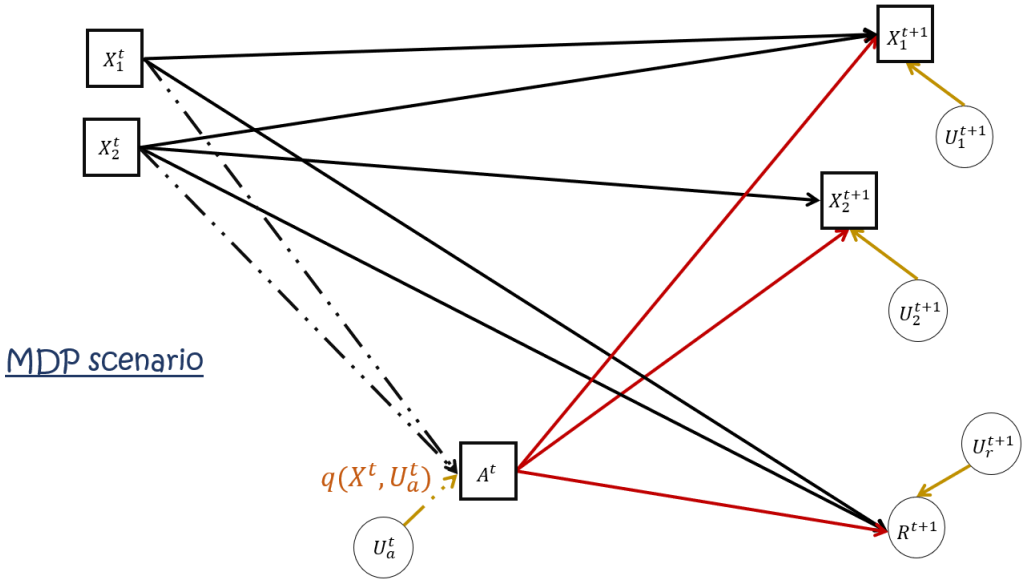


FIGURE 7.6 – Temporal causal graph for toy example

7.2.5.2 MDP counterfactual

In the MDP scenario, since all endogenous variables are observed, the counterfactual reasoning can be utilised to infer information about exogenous noise variables U, i.e. about the system's randomness similarly as the learning of stochastic transitions in the state-transitional model. This is the first step of counterfactual reasoning, corresponding to the *abduction* phase and we will show the process for one single variable s_i.

Abduction In general, the causal assignment associated to a variable s_i takes as input parent's variables at time t of the local variable s_i, the agent's decision a^t and a noise u_i^t, then outputs the value of s_i at time $t+1$.

$$s_i^{t+1} = f_i(Pa(s_i^{t+1}), a^t, u_i^{t+1})$$

For a given tuple $(Pa(s_i^{t+1}), a^t, s_i^{t+1})$, obtained from data, the agent can infer the noise value u_i^{t+1} by taking the inverse of the causal assignment f_i under some assumptions on the function form :

$$u_i^{t+1} = f_i^{-1}(s_i^{t+1}, Pa(s_i^{t+1}), a^t)$$

We denote by $\bar{u}_i$ the discovered noise value. Therefore the agent has now the *evidence* of the system at time t when he did action a^t and can infer the outcome of non taken decisions $a' \in \mathcal{A} \setminus \{a\}$, by doing the *intervention* and the *prediction* phases with its updated SCM. Concretely in our example, its new SCM would be :

$$\mathcal{M}_{MDP} = \begin{cases} A^t = q(X_1^t, X_2^t, \bar{u_a}) \\ X_1^{t+1} = f_1(X_1^t, X_2^t, A^t, \bar{u_1}) \\ X_2^{t+1} = f_2(X_2^t, A^t, \bar{u_2}) \\ \mathcal{R}^{t+1} = f_r(A^t, X_1^t, X_2^t, \bar{u_r}) \end{cases}$$

Intervention With its updated SCM, the learning agent can imagine a new intervention (the counterfactual decision) in the system by doing action a' in the system $do(A^t = a')$. In our example, this would change the SCM as follows :

$$\mathcal{M}_{MDP} = \begin{cases} a' \leftarrow do(A^t = a') \\ X_1^{t+1} = f_1(X_1^t, X_2^t, a', \bar{u_1}) \\ X_2^{t+1} = f_2(X_2^t, a', \bar{u_2}) \\ \mathcal{R}^{t+1} = f_r(a', X_1^t, X_2^t, \bar{u_r}) \end{cases}$$

Prediction Finally, it can predict or evaluate the outcome of the system by using causal assignments and the evidence to predict what would action a' do :

$$s_i^{t+1} = f_i(Pa(s_i^{t+1}), a', \bar{u}_i)$$

In our example, this would mean to evaluate X_1^{t+1}, X_2^{t+1} and r^{t+1} from the SCM computations.

Notice that we have shown the counterfactual process on one data sample by discovering the value of noises and using these evidence values to predict new outcomes. This can be generalised by considering distribution of noises, i.e. the agent has an a priori distribution for noise variables $P_{prior}(U)$ which can be updated from the noise values discovery with Bayesian learning techniques, leading to $P_{posterior}(U)$. Finally, the evaluation of non taken decisions is made by simulating the system with the updated SCM that includes the a posteriori distributions of noise variables. This allows for a broader and more generic evaluation of counterfactuals since it considers larger possibilities for the noise values.

Finally, we depict in Algorithm 30 a causal Dyna-Q-Learning method with the counterfactual reasoning to discover noise values and do planning by evaluating counterfactuals. This method can be seen as a data augmentation technique since the agent will be able to plan and update its Q-function for all possible actions at each time step. At each time step, when the agent performs action a in state s, it recovers from the SCM and the abduction phase the noise values of the system (We denote in the pseudocode this process by *InferenceSCM*). Next it updates the Q-function for all the non taken actions with the SCM.

Algorithm 30: Counterfactual-based data augmentation RL (*CDYNA - Causal Dyna-Q-Learning*)

Input: Q_0, q_0, α learning rate, γ discount rate, ϵ
Output: q^*, Q^*
Data: SCM $\mathcal{M}$ is known, $\mathcal{P}(\mathcal{U})$ is unknown

```
/* Loop until end of episodes or criterion                                   */
for e ∈ MaxEpisode do
    Select state s ∈ S , s = (s_1, ..., s_n)                  // Initial state
    for i ∈ MaxIteration do
        Take action a ∈ A with ε-greedy policy                // Action selection
        Observe s' and reward r(s,a) and collect tuple < s,a,r,s' >   // Collecting tuples data
        Q(s,a) ← Q(s,a) + α [r(s,a) + γ max'_a Q(s',a') − Q(s,a)]     // Update Q
        Ū_i ← InferenceSCM(s,a,r,s')                          // Abduction
        /* Counterfactual evaluation of non-chosen actions                   */
        for a' ∈ A \ a do
            Replace action a' in the SCM                      // Intervention
            Calculate r', s'' = F(s,a',Ū_i)                   // Prediction
            Q(s,a') ← Q(s,a') + α [r' + γ max_a'' Q(s'',a'') − Q(s,a)]   // Update Q
```

7.2.5.3 POMDP counterfactual

Let us now consider the partially observable scenario with no confounding variable, i.e. it exists some endogenous variables in V that are unobserved and the Markovian property is violated. In this context, we should treat the problem by using history of data to gain information about transitions between two time steps. Suppose that in our example the endogenous variable X_2 is unobserved. This leads to the following SCM that the agent carries where the agent's policy is only influenced by variable X_1 :

$$\mathcal{M}_{POMDP} = \begin{cases} A^t = q(X_1^t, u_a^t) \\ X_1^{t+1} = f_1(X_1^t, X_2^t, A^t, u_1^{t+1}) \\ \left(X_2^{t+1} = f_2(X_2^t, A^t, u_2^{t+1})\right) \text{ unobserved} \\ R^{t+1} = f_r(A^t, X_1^t, X_2^t, U_r^{t+1}) \end{cases}$$

We display the partially observable example in Figure 7.7.

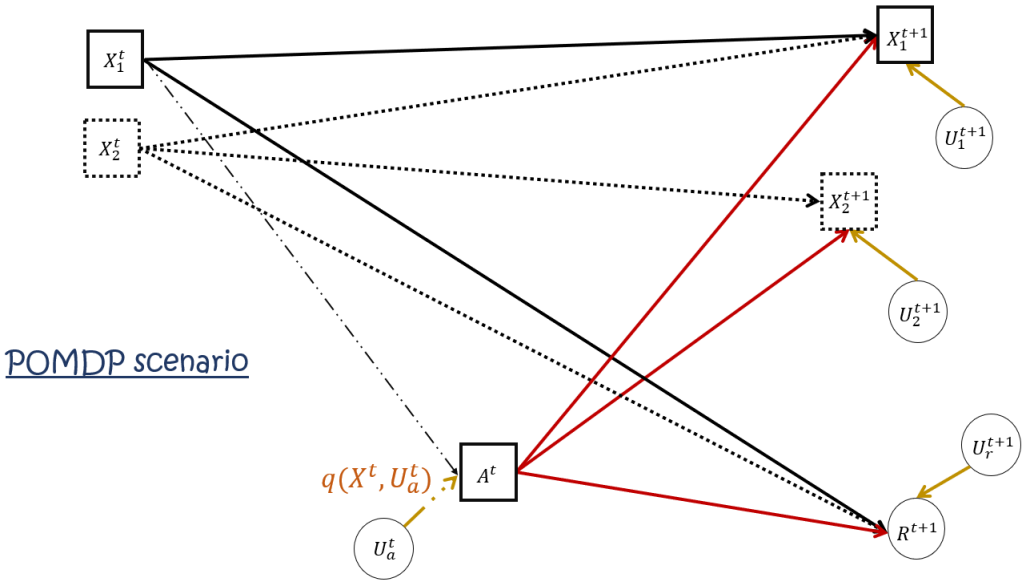


FIGURE 7.7 – Temporal causal graph for toy example with partial observation

The counterfactual reasoning in the POMDP scenario should integrate history of data and should discover not only exogenous noise variables but also hidden variables. In usual POMDP solutions, there exist algorithms that deal with a belief function b of the real state when observing o, i.e. $b(o)$ gives the probability to be in states s. The Algorithm 31 proposed by Deepmind's team in [30] has a similar idea where the abduction phase ought to learn a conditional distribution of hidden variables from history of observations. Intuitively, the inference of hidden variables made with the SCM and with collected data should act as a proxy for information on past transitions in the system. The first step for the reinforcement learner is to collect interventional data. In these data, only endogenous observed values are displayed (a, r, s_{obs}). We define by $\hat{D} = \{ a^t, r^{t+1}, s_{obs}^t, s_{obs}^{t+1} \}$ the evidences obtained by interventions in the environment for one iteration. We want to infer hidden variables given $\hat{D}$ with the SCM then evaluate counterfactual. To do so we use causal assignments F inverse functions to compute the values for unobserved variables, in the same setting such as in the MDP scenario for noise values. This allows the learning agent to infer conditional distributions of hidden variables and noises, conditioned on the history of data : $P(U|\hat{D})$ and $P(s_i|\hat{D})$. These conditional distributions are next integrated in the SCM for *intervention* and *prediction* phases.

7.2.5.4 MDPUC counterfactual

Last scenario focuses on MDP environments with unobserved confounder. In this setting, as we have seen in Section 7.2.4, the unobserved confounder does not violate the Markovian property since the agent still has the necessary information to understand how the system evolves between two time steps t and $t+1$. Following the same example as POMDP, in scenarios with unobserved confounders, we assume the agent can not measure the variable X_2 but this variable follows a distribution on its own. This changes the SCM as follows :

$$\mathcal{M}_{MDPUC} = \begin{cases} A^t = q(X_1^t, u_a^t) \\ X_1^{t+1} = f_1(X_1^t, X_2^t, A^t, u_1^{t+1}) \\ \left(X_2^{t+1} \sim \mathbb{P}_2\right) \text{ unobserved} \\ R^{t+1} = f_r(A^t, X_1^t, X_2^t, U_r^{t+1}) \end{cases}$$

We display the unobserved confounder case in Figure 7.8.

Algorithm 31: Causal policy-based reinforcement learning for POMDP [30]

Input: $q^0 = \mu^0$, structural causal model $\mathcal{M} : \forall i \in |\mathcal{S}|, \forall a \in \mathcal{A}, s_i := f_i(Pa(s_i)) + \mathcal{U}_i$, N : frequency update

Output: q^*

Data: Causal graph $\mathcal{G}$, structural assignments F are known, Noise distributions $\mathcal{P}(\mathcal{U})$ is unknown

for $e \in MaxEpisode$ **do**
  Select state $s \in \mathcal{S}$, $s = (s_1, \ldots, s_k)$ // Initial state
  /* Step 1: Collect interventional data $\mathcal{D}_{obs}$ running policy q in the environment */
  for $i \in MaxIteration$ **do**
    Take action $a \in \mathcal{A}$ following $q(s, u_A)$. // Action selection
    Observe s' and reward r and collect tuple $< s, a, r, s' >$ in $\mathcal{D}_{obs}$ // Collecting tuples data
  /* Step 2: Infer noises distribution from $\mathcal{D}_{obs}$ */
  Knowing F, infer $\mathcal{P}(\mathcal{U}|\mathcal{D}_{obs})$ // Abduction
  Replace prior $\mathcal{P}(\mathcal{U}) \leftarrow \mathcal{P}(\mathcal{U}|\mathcal{D}_{obs})$ in model $\mathcal{M}$
  /* Step 3: Digital twin simulation for counterfactual evaluation */
  Simulate several trajectories with digital twin model $\mathcal{M}$ with F and $\mathcal{P}(\mathcal{U})$ by running policy μ
  /* Step 4: Policy update */
  Update policy μ with trajectories returns
  /* Step 5: Push new policy in real environment */
  if $e\%N == 0$ **then**
    $q \leftarrow \mu$

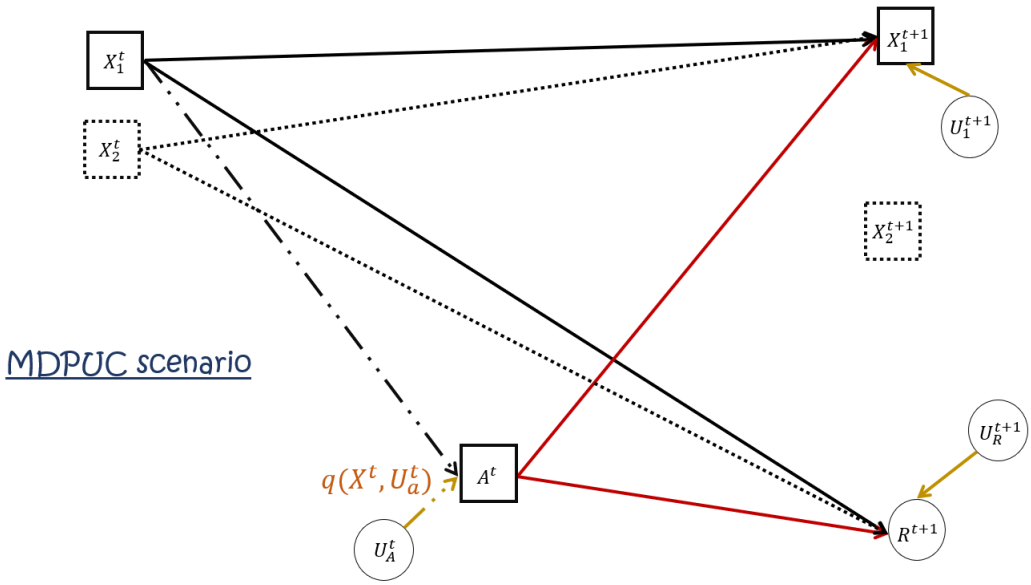


FIGURE 7.8 – Temporal causal graph for toy example with unobserved confounder

In the MDPUC context, there is no need to treat the problem with historical data since the Markovian property is not violated. Therefore, the counterfactual reasoning process here would only infer an a posteriori distribution of the hidden variable X_2, not conditioned on history of data. Again, this learned distribution would serve as an element of the new SCM to generate rollouts for counterfactual planning. The algorithm that we will use in the experimental results Section 7.4 in the Algorithm 30 but the InferenceSCM() part will also discover the value of the UC variable.

7.2.5.5 Overview of counterfactual in MDP environments

To summarise the counterfactual reasoning in Reinforcement Learning, the learning agent is provided the SCM of the world (relations between environment variables, causal assignments) and can infer information about hidden and noise variables from collected interventional data and by taking the inverse of causal functions. Depending on the scenario (POMDP or MPDUC), the agent will deal with historical data or not. Finally, it will evaluate counterfactual decisions based on its updated SCM with distributions or discovered values for the hidden variables. Notice that this counterfactual planning is different than initial planning of model-based RL techniques. The simulated trajectories are generated by differentiating the causal deterministic part and the randomness part of the system with noises distributions. This creates more precise trajectories illustrated with the following example.

Example 5. *Planning in Model-based RL : The agent will generate rollouts with its model $\mathcal{P}$ and $\mathcal{R}$ such that $s', r \leftarrow \mathcal{P}(s', s, a), \mathcal{R}(s, a)$ and will update its policy or value function according to this sample (or trajectory). Yet, the evidence of the environment that occurred when the agent visited (s, a) in the past are lost in the state-transitional model because there is no distinction between causal assignments and randomness.*

Planning in Counterfactual RL : The agent will generate rollouts with its SCM model and inferred distributions of hidden and noise variables such that $s', r \leftarrow F(s, a, u), P(U), \mathcal{R}(s, a, u)$. This sampling process is more precise because it considers the evidence information that occurred when the agent has visited the state-action pair (s, a) in the past.

In a nutshell, the counterfactual process should provide a gain in usual MDP systems by accelerating the convergence because the counterfactual process could evaluate multiple non-taken decisions more quickly. However the main benefit will reside in context with hidden variables where non causal models will suffer to infer information about unobserved variables and can be potentially biased resulting in learned policies with poor quality. This is what we will demonstrate in Section 7.4 with an application for the tandem queue Cloud environment.

7.3 Causal modelling of multi-tier Cloud architectures

This section presents the application of CRL techniques in multi-tier Cloud architectures. It consists of reformalising the MDP model with SCM modelling and comparing counterfactual reinforcement learning methods with SoTA RL algorithms in different environments (MDP, POMDP, MPDUC). However, we let the treatment of POMDP cases in perspective and only deal with the initial tandem queue environment and scenario with unobserved confounders, namely MDP and MDPUC environments.

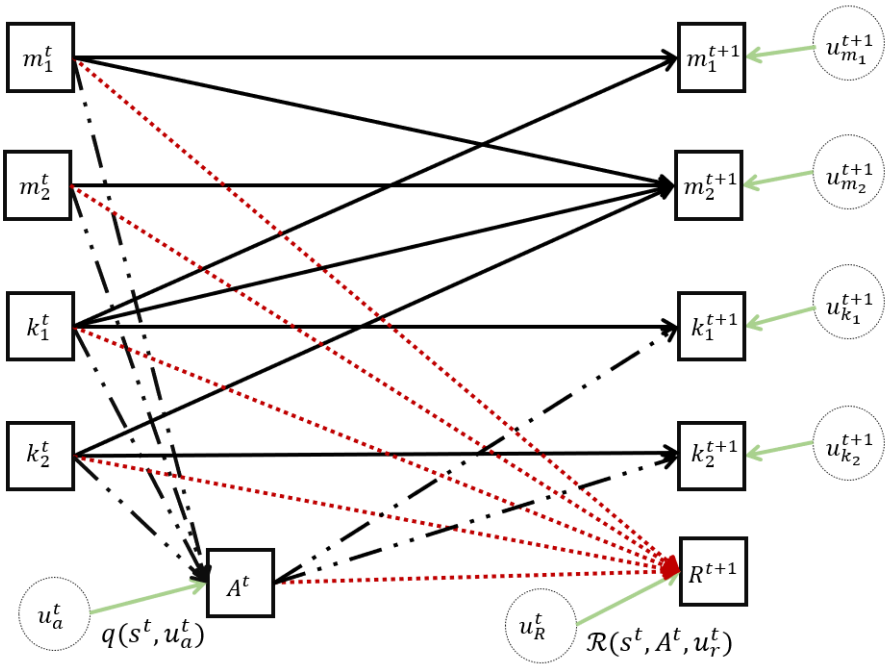


FIGURE 7.9 – Tandem queue temporal causal graph

7.3.1 $\mathcal{E}nv\ \mathcal{A}_1$ Tandem queue MDP model

7.3.1.1 SCM representation

We describe the queuing system under an SCM representation which is equivalent to the initial state-transitional model presented in Chapter V with $\mathcal{P}$ and $\mathcal{R}$. We depict in Figure 7.9 the causal graph associated to the SCM representation of the tandem queue scenario. Black dot lines represent the influence of state variables s_i on the policy q, thus on the agent's decision a^t at time t. Black lines represent the local dependencies between state variables between time t and time $t+1$. The red dot lines represent the influences of state variables and action on the reward. Finally, green lines show effects of noise variables on environment variables. Noise variables $\mathcal{U}$ represents the randomness of arrivals u_{m_1} in node 1, potential breakouts in node 1 u_{k_1}, in node 2 u_{k_2}, and the randomness of departure from node 1 to node 2 with u_{m_2}. The variable u_A^t represents the randomness if the policy q is stochastic but for convenience we will set $u_A^t = 0$ since we only consider deterministic policies in this document. U_r^t represents the reward noise. In $\mathcal{E}$nv 1, noises U_{k_1} and U_{k_2} are set to 0 since we assume no breakouts can occur, i.e. the influence of agent's decision on the number of activated virtual resources is fully deterministic. Same reasoning occurs for the reward function which is deterministic in this environment.

Notice that the Figure 7.9 shows the behaviour of the system when the agent follows its policy q. Yet in the reinforcement learning paradigm (layer 2 of causality), it will perform interventions on the system by acting on variable A^t. The agent will intervene by changing value of A^t with $do(A^t = (a_1, a_2))$. We depict in Figure 7.10 the behavior of the system when the agent takes decisions different from the policy for exploration. In this setting, the influence of other variables are removed.

Next, we provide the equivalent SCM representation of the tandem queue MDP environment providing the causal assignment :

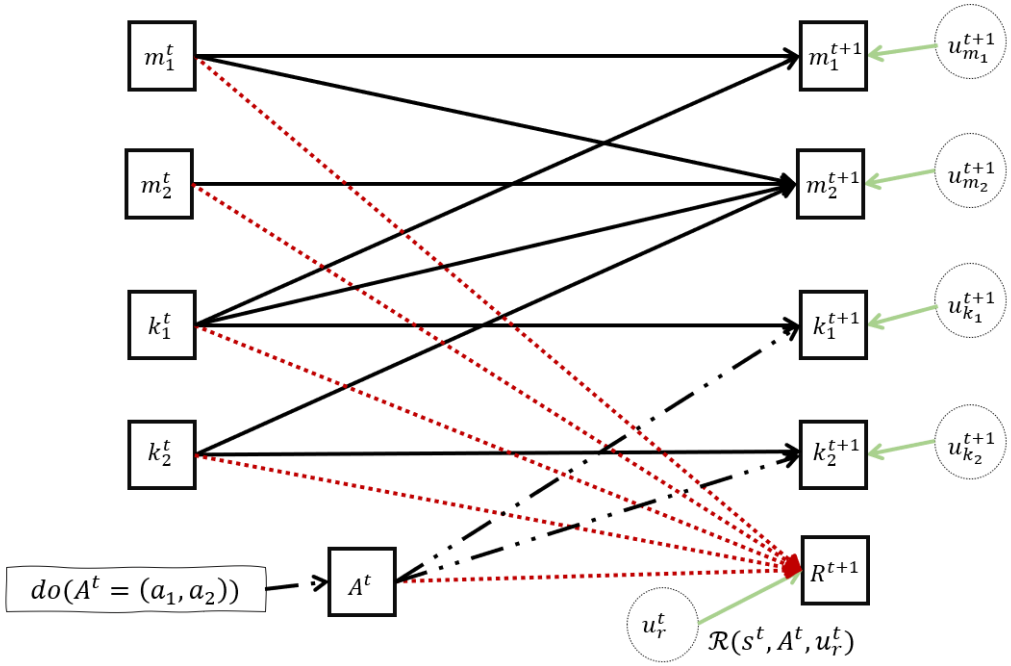


FIGURE 7.10 – Tandem queue temporal causal graph under intervention

$$\mathcal{M} = \begin{cases} A^t = q(m_1^t, k_1^t, m_2^t, k_2^t, U_a^t) \\ m_1^{t+1} = f_{m1}(m_1^t, k_1^t, A^t, U_{m_1}^t) \\ k_1^{t+1} = f_{k1}(k_1^t, A^t, U_{k_1}^t) \\ m_2^{t+1} = f_{m2}(m_1^t, m_2^t, A^t, k_1^t, k_2^t, U_{m_2}^t) \\ k_2^{t+1} = f_{k2}(k_2^t, A^t, U_{k_2}^t) \\ R^{t+1} = \mathcal{R}(m_1^t, k_1^t, m_2^t, k_2^t, , A^t, U_r^t) \end{cases}$$

Since we assume that $U_a^t = U_r^t = U_{k_1}^t = U_{k_2}^t = 0$, we have :

$$\mathcal{M} = \begin{cases} A^t = (a_1^t, a_2^t) = q(m_1^t, k_1^t, m_2^t, k_2^t) \\ m_1^{t+1} = f_{m1}(m_1^t, k_1^t, A^t, U_{m_1}^t) \\ k_1^{t+1} = k_1^t + a_1^t \\ m_2^{t+1} = f_{m2}(m_1^t, m_2^t, A^t, k_1^t, k_2^t, U_{m_2}^t) \\ k_2^{t+1} = k_2^t + a_2^t \\ R^{t+1} = \mathcal{R}(m_1^t, k_1^t, m_2^t, k_2^t, A^t) \end{cases}$$

Since $\mathcal{R}$ is the same reward function defined in Chapter V Section 5.2.1, we only need to find mathematical expression of functions f_{m_1} and f_{m_2} to have an equivalent representation for the tandem queue environment.

Let us recall the transitions for state variables m_1^{t+1} and m_2^{t+1}.

$$m_1^{t+1} = \begin{cases} m_1^t + 1 \text{ if arrival in } node1; \\ m_1^t - 1 \text{ if departure from } node1; \\ m_1^t \text{ else.} \end{cases} \quad \text{and} \quad m_2^{t+1} = \begin{cases} m_2^t + 1 \text{ if arrival in } node2; \\ m_2^t - 1 \text{ if departure from } node2; \\ m_2^t \text{ else.} \end{cases} \quad \text{(M1M2)}$$

Therefore we have the following probability transitions :

$$P(m_1^{t+1}|Pa(m_1), a^t) = \begin{cases} \frac{\lambda}{\tilde{\Lambda}} \text{ if arrival in } N_1; \\ \frac{\mu_1(k_1^t + a_1^t)}{\tilde{\Lambda}} \text{ if departure from } N_1; \\ \frac{\tilde{\Lambda} - \lambda - \mu_1(k_1^t + a_1^t)}{\tilde{\Lambda}} \text{ else.} \end{cases}$$

and

$$P(m_2^{t+1}|Pa(m_2), a^t) = \begin{cases} \frac{\mu_1(k_1^t + a_1^t)}{\tilde{\Lambda}} \text{ if arrival in } N_2; \\ \frac{\mu_2(k_2^t + a_2^t)}{\tilde{\Lambda}} \text{ if departure from } N_2; \\ \frac{\tilde{\Lambda} - \mu_1(k_1^t + a_1^t) - \mu_2(k_2^t + a_2^t)}{\tilde{\Lambda}} \text{ else.} \end{cases}$$

We propose to rewrite the following dynamics in the SCM form, i.e. with causal assignments to express what will be the next value of state variables at time $t+1$. (M1M2) can be rewritten by :

$m_1^{t+1} = m_1^t + \mathbb{1}_{\text{(event=arrival)}} - \mathbb{1}_{\text{(event=departure)}}$

$\Leftrightarrow m_1^{t+1} = m_1^t + \mathbb{1}_{U_{m_1} \leq \lambda/\tilde{\Lambda}} - \mathbb{1}_{U_{m_1} \leq (\lambda + \mu_1(k_1^t + a_1^t))/\tilde{\Lambda}}$ where $U_{m_1}^t \sim Unif[0,1]$

and

$m_2^{t+1} = m_2^t + \mathbb{1}_{U_{m_2}^t \leq \lambda/\tilde{\Lambda}} - \mathbb{1}_{U_{m_2}^t \leq (\lambda + \mu_1(k_1^t + a_1^t))/\tilde{\Lambda}}$ where $U_{m_2}^t \sim Unif[0,1]$

Finally, we have the SCM representing tandem queue dynamics with :

$$\mathcal{M} = \begin{cases} A^t = q(m_1^t, k_1^t, m_2^t, k_2^t) \\ m_1^{t+1} = m_1^t + \mathbb{1}_{U_{m_1}^{t+1} \leq \lambda/\tilde{\Lambda}} - \mathbb{1}_{U_{m_1} \leq (\lambda + \mu_1(k_1^t + a_1^t))/\tilde{\Lambda}} \text{ where } U_{m_1}^{t+1} \sim Unif[0,1] \\ k_1^{t+1} = k_1^t + a_1^t \\ m_2^{t+1} = m_2^t + \mathbb{1}_{U_{m_1}^{t+1} \leq (\lambda + \mu_1(k_1^t + a_1^t))/\tilde{\Lambda}} - \mathbb{1}_{U_{m_2}^{t+1} \leq (\mu_1(k_1^t + a_1^t)) + \mu_2(k_2^t + a_2^t))/\tilde{\Lambda}} \text{ where } U_{m_2}^{t+1} \sim Unif[0,1] \\ k_2^{t+1} = k_2^t + a_2^t \\ R^{t+1} = \mathcal{R}(m_1^t, k_1^t, m_2^t, k_2^t, A^t) \end{cases}$$

Notice that we could also have found the functions with the inverse-CDF method and the proposed method from the Deepmind's work [30] in Lemma 5.

7.3.1.2 Counterfactual calculations

Recall that we consider the MDP scenario first where the agent can observe the full environment state. According to Section 7.2.5, it only requires to discover the exogenous noise values with the counterfactual reasoning. Thus, for one experience tuple, the agent can discover the noises realisations with inverse causal assignments in the following way :

$$\mathcal{U} = \begin{cases} U_{m_1}^{t+1} = f_{m1}^{-1}(m_1^t, k_1^t, A^t, m_1^{t+1}) \\ U_{m_2}^{t+1} = f_{m2}^{-1}(m_1^t, m_2^t, A^t, k_2^t, m_2^{t+1}) \end{cases}$$

Concretely, from a data sample (s, a, r, s') it computes from the SCM the noise values $\bar{u}_{m_1}$ and $\bar{u}_{m_2}$:

$$\mathcal{U} = \begin{cases} \bar{u}_{m_1} = f_{m1}^{-1}(m_1, k_1 + a_1, m_1') \\ \bar{u}_{m_2} = f_{m2}^{-1}(m_1, m_2, k_1 + a_1, k_2 + a_2, m_2') \end{cases}$$

Once the agent has the noise realisations under a specific experience tuple, it can evaluate counterfactual decisions following Algorithm 30. As in Chapter V and Chapter VI the learning agent observes from the environment samples (s, a, r, s') and ought to learn the dynamics of the system by discovering exogenous noise variables. We

will compare the causal RL approach with Q-Learning, Dyna-Q and MDP online technique on the tandem queue scenario in Section 7.4.

7.3.2 $\mathcal{E}nv\,\mathcal{C}_1$ Tandem queue MDPUC environment

To present the MDPUC scenario, we first provide more details about the scenario considered by Bareinboim et al. in [153]. Next, we display the SCM representation of the tandem queue with unobserved confounder and finally provide experimental results with the causal RL algorithms.

7.3.2.1 Toy example illustration [153]

We present here the reproduction of the results of Bareinboim et al. in [153] that allows us to consider the MDPUC scenario for the tandem queue environment and that will help for understanding. In their paper, the authors consider a medical treatment example where a physician has to heal patients at the hospital by providing a treatment or not $A = \{0, 1\}$. The patient has a given level of corticosteroids $C = \{0, 1\}$ and a given health which is the outcome to maximise here : $R = \{0, 1\}$. The goal of the learning agent is to maximise the cumulative health score in the long term with a discounted rate $\gamma = 0.99$. Moreover, the agent's health score R is influenced by external factors such as economical status E and patient's mood M which are also binary variables. Last, the physician's decision is influenced by the external factors that are observed from him when he treats the patients. However these variables have a confounding effect since they also influence the health of the patient. We depict in Figure 7.11 the behavior of the environment when the physician follows its policy q_{phy}.

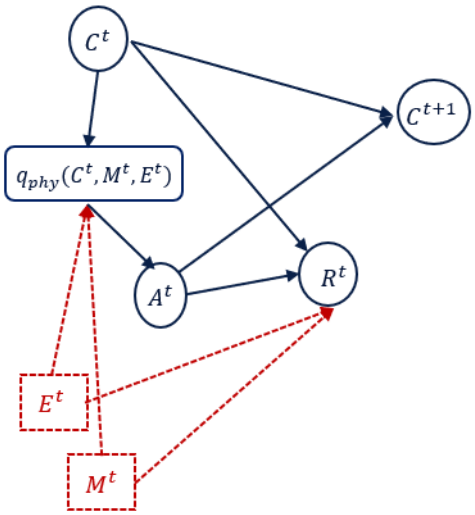


FIGURE 7.11 – Environment behavior under physician's policy q

The goal of this paper is to demonstrate the gain of counterfactual reasoning in reinforcement learning. The authors settle a scenario where a learning agent has to improve the physician's policy from collected data but the variables E and M were not stored in the data. This is a fair assumption in real environments since industrial applications could require to find better solutions than existing one and the existing solutions conducted by human experts could be biased by confounding factors that are not measurable in the data (e.g. autonomous vehicles with imitation learning). Therefore in their example, the learning agent only has observational data of C, R and $a = q_{phy}()$. It acts by changing the value of A and observe the outcome R. The objective of the software agent is to improve $q_{phy}()$ by providing a better policy $q_{ctf}()$. This is shown in Figure 7.12.

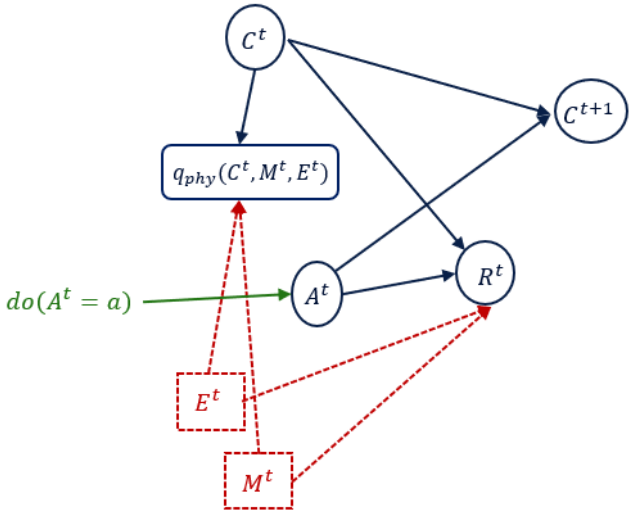


FIGURE 7.12 – Environment behavior under the learning agent's decisions

The method proposed by Bareinboim et al. is to extend the observation space of the learning agent with the expert's policy that acts as a proxy to the confounding variables M and E. In this setting, the *abduction* phase of the counterfactual reasoning is to infer from the physician's decision what was the value of variables M and E when the expert took his decision. This is depicted in Figure 7.13.

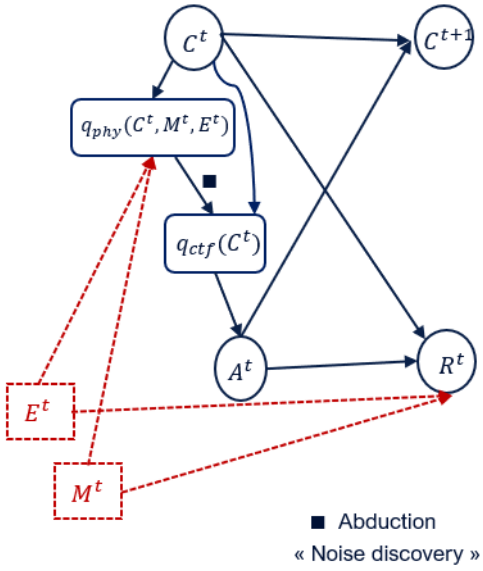


FIGURE 7.13 – Environment behavior under the learning agent's decisions with the extended observation

To fully grasp the counterfactual reasoning in a MDPUC scenario, we reproduced the simulation proposed by the authors [153]. It compares four algorithms/policies under environments characteristics, i.e. how the system evolves. We did the same process for one of the environment in which the expert's policy is very bad. The comparison is done offline between a random policy, the physician's policy, an oracle/optimal policy and two policies that have been learned by the software agents with MDP online techniques. The first method called *Mormax* [136] is very close to Rmax and the second one is a causal version of Mormax with the extended observation and counterfactual reasoning. The numerical experiments displayed in Figures 7.14 and 7.15 show the cumulative reward and

average reward obtained from the different policies when simulating the environment. Notice that the scenario we selected from the paper is a case where the physician's policy is very bad. Moreover, we can see that a learning agent (Mormax) does not perform better than a random policy showing how hidden confounding variables can deteriorate policies quality. Last the results demonstrate a gain using counterfactual reasoning in RL (Mormax Causal) algorithm.

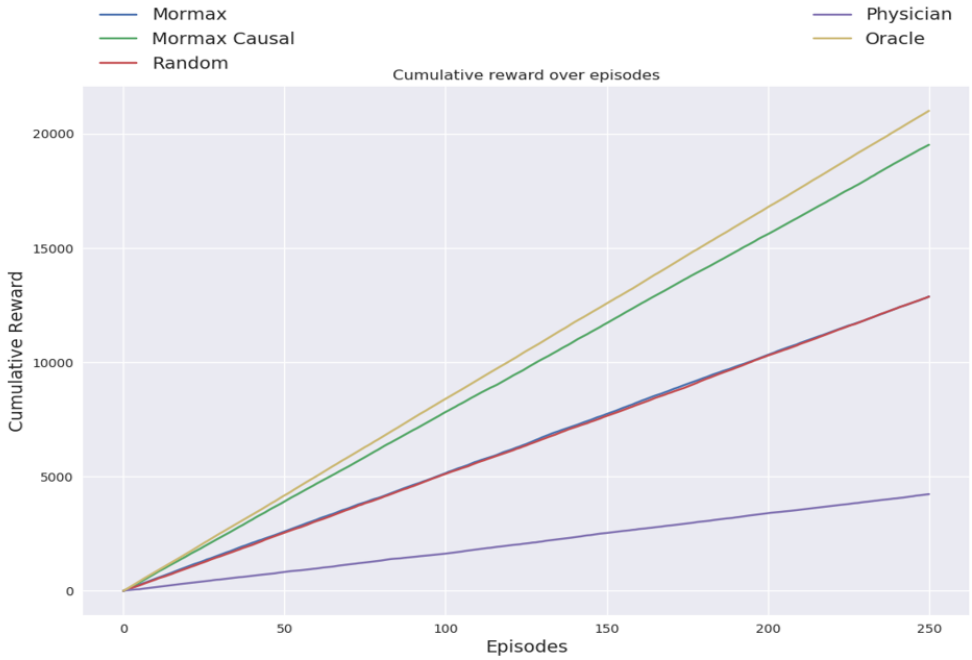


FIGURE 7.14 – Cumulative reward obtained by different algorithms over learning episodes

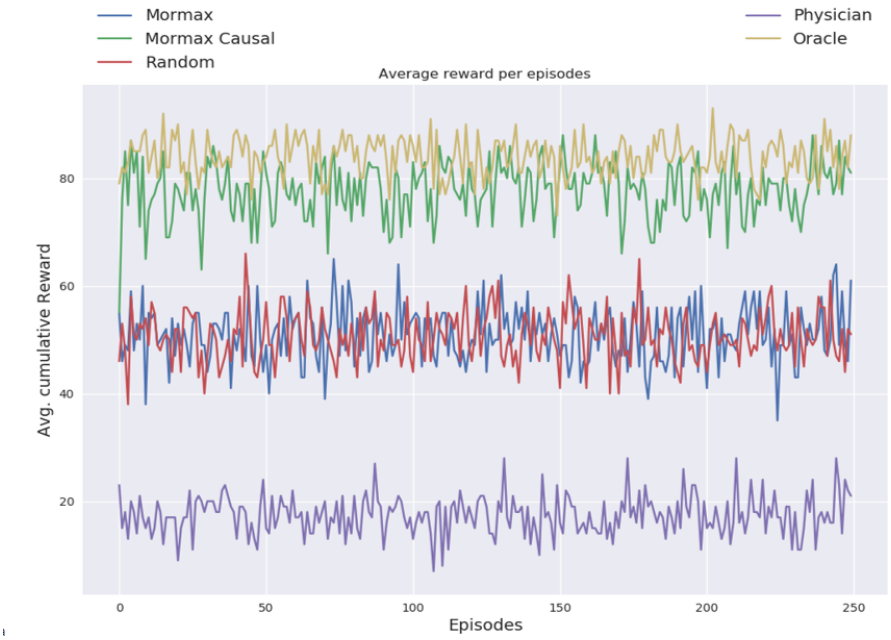


FIGURE 7.15 – Average reward obtained by different algorithms over learning episodes

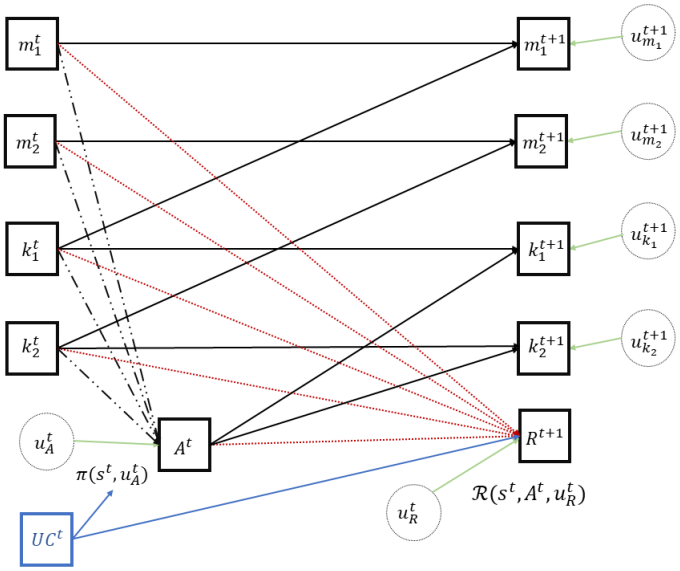


FIGURE 7.16 – MDPUC Tandem queue temporal causal graph

7.3.2.2 SCM representation

To come back to our use-case, our goal was to consider a similar scenario in the tandem queue environment so we can demonstrate the gain of counterfactual reasoning in RL methods. This is why we consider in the last part the unobserved confounder scenario where there might exist unobserved variable that confound environment variables and an expert's policy. This environment is derived from the MMPP tandem queue environment described in Chapter V Section 5.2.3.1. However, the MMPP scenario is a POMDP environment therefore we slightly modify this environment to fall in the MDPUC setting. For this purpose, the arrival rate λ is still hidden but follows a distribution on its own. In other words, its value depends on a probability distribution $\mathbb{P}_\lambda$ and does not depend on previous values of arrival rates, which is still consistent with real Cloud systems. Therefore, at each time step, the variable λ is drawn from this distribution, no matter what was its previous value. The environment state is given by $s = (\lambda, m_1, m_2, k_1, k_2)$ and the agent only observes $o = (m_1, m_2, k_1, k_2)$. We depict in Figure 7.16 the MDPUC environment when the agent follows its policy q and in Figure 7.17 when the agent does interventions.

The associated structural causal model is described below with the same consideration for noise variables as in the fully observable MDP scenario $(U_r = U_{k_1} = U_{k_2} = U_a = 0)$:

$$\mathcal{M} = \begin{cases} A^t = q(m_1^t, m_2^t, k_1^t, k_2^t) \\ \lambda^t = U_\lambda \text{ where } U_\lambda \sim \mathbb{P}_\lambda \\ m_1^{t+1} = f_{m1}(m_1^t, k_1^t, A^t, \lambda^t, U_{m1}^{t+1}) \\ k_1^{t+1} = f_{k1}(k_1^t, A^t) \\ m_2^{t+1} = f_{m2}(m_1^t, m_2^t, A^t, \lambda^t, k_1^t, k_2^t, U_{m2}^{t+1}) \\ k_2^{t+1} = f_{k2}(k_2^t, A^t) \\ R^{t+1} = \mathcal{R}^t(m_1^t, k_1^t, m_2^t, k_2^t, A^t) \end{cases}$$

In detail we have :

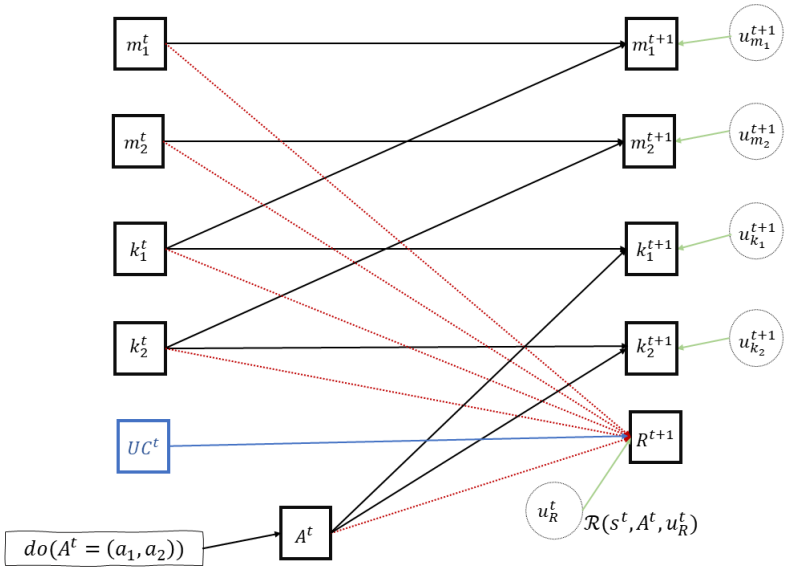


FIGURE 7.17 – MDPUC Tandem queue temporal causal graph under intervention

$$\mathcal{M} = \begin{cases} A^t = q(m_1^t, k_1^t, m_2^t, k_2^t, U_A^t) \\ \lambda^t = U_\lambda \text{ where } U_\lambda \sim \mathbb{P}_\lambda \text{ (unobserved)} \\ m_1^{t+1} = m_1^t + \mathbb{1}_{U_{m1}^{t+1} \leq \lambda^t / \tilde{\Lambda}} - \mathbb{1}_{U_{m1}^{t+1} \leq (\lambda^t + \mu_1(k_1^t + a_1^t))/\tilde{\Lambda}} \text{ where } U_{m_1}^t \sim Unif[0,1] \\ k_1^{t+1} = k_1^t + a_1^t \\ m_2^{t+1} = m_2^t + \mathbb{1}_{U_{m1}^{t+1} \leq (\lambda^t + \mu_1(k_1^t + a_1^t))/\tilde{\Lambda}} - \mathbb{1}_{U_{m2}^{t+1} \leq (\mu_1(k_1^t + a_1^t)) + \mu_2(k_2^t + a_2^t))/\tilde{\Lambda}} \text{ where } U_{m_2}^{t+1} \sim Unif[0,1] \\ k_2^{t+1} = k_2^t + a_2^t \\ R^{t+1} = \mathcal{R}(m_1^t, k_1^t, m_2^t, k_2^t, , A^t, Z^t, U_R^t) \end{cases}$$

To remain faithful to the scenario of [153], we consider in our use-case that a human is acting in the system according to a policy q_{hum} and imagines possible values of λ by observing the system, i.e. it derives from the values of m_1 and m_2, a possible value of λ. This process remains consistent with real Cloud environments. Therefore, our goal is to improve a human's policy with a learning software agent that will learn a policy with RL methods on collected data, similarly as in [153].

7.3.2.3 Counterfactual calculations

The counterfactual reasoning in this situation would be to infer noise variables, similarly as in the MDP scenario but also the hidden arrival rate variable λ. For this purpose, we augment the observation space of the learning agent with the human's decision taken in the data. Therefore we consider the following observations $o = (m_1, m_2, k_1, k_2, q_{hum})$ where q_{hum} acts as a proxy for confounding variables and belongs to the *abduction* phase. The noise variables u_{m_1} and u_{m_2} are discovered with the same computations as in the MDP scenario. Finally, the *intervention* and *prediction* phase remains the same. Henceforth, we will also use the Algorithm 30 only it will be improved with the augmented observation proposed by [153].

7.4 Experimental results

In this section we evaluate counterfactual RL algorithms compared with state of the art methods that were assessed in previous chapters. Moreover we consider only countable state space scenarios, i.e. $\mathcal{A}_1, \mathcal{C}_1$ environments.

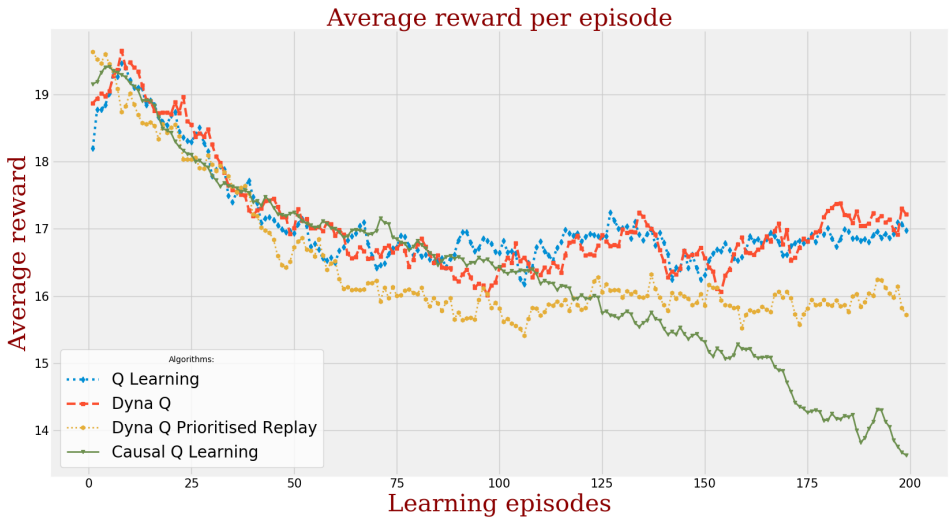


FIGURE 7.18 – Average reward over learning episodes between RL algorithms in MDP scenario

Since we do not deal with the POMDP scenario, our counterfactual RL method does not provide a solution considering history of observations. Therefore our goal in this section is to compare the *counterfactual Q-Learning algorithm* described in Algorithm 30 with Q-Learning and Dyna-Q-Learning methods.

7.4.1 Environments and Simulation Parameters

We keep the same python Gym simulator as described in Chapter V Section 5.4. In this chapter, we extend also the Gym simulator with two cases, the MMPP scenario for POMDP use-case and the MDPUC scenario, basically where the arrival rate is following a distribution on its own.

7.4.2 $\mathcal{E}nv\ \mathcal{A}_1$ Tandem queue MDP environment

We display in Figure 7.18 the average reward over learning episodes obtained during the learning process for different algorithms on the chosen Cloud scenario. For fair comparison, we have parameterised the Dyna-Q method such that it does only 8 one-step planning phase at each iteration such that the counterfactual RL method which will evaluate the 8 other counterfactual decisions. We show a gain in the speed of convergence with the causal RL method compared to Q-Learning and to Dyna-Q which means that the counterfactual evaluation helps to accelerate the convergence (same complexity as Dyna-Q).

7.4.3 $\mathcal{E}nv\ \mathcal{C}_1$ Tandem queue MDPUC environment

We display in Figure 7.19 the average reward over learning episodes obtained during the learning process for the MDPUC scenario. The arrival rate λ was taking several values and changing at a given frequency, where its values were drawn from a uniform distribution. We first observe that the Q-learning algorithm was suffering to learn an efficient policy due to unobserved arrival rate that was varying many times during the learning. On the other hand, the Causal Q Learning could handle more efficiently this scenario with the augmented state space where the new feature was a predefined policy of an 'expert'. This policy was telling the 'expert' what action to play with a function telling him what was the possible value of λ in a given observation of the system (m_1, m_2, k_1, k_2). This

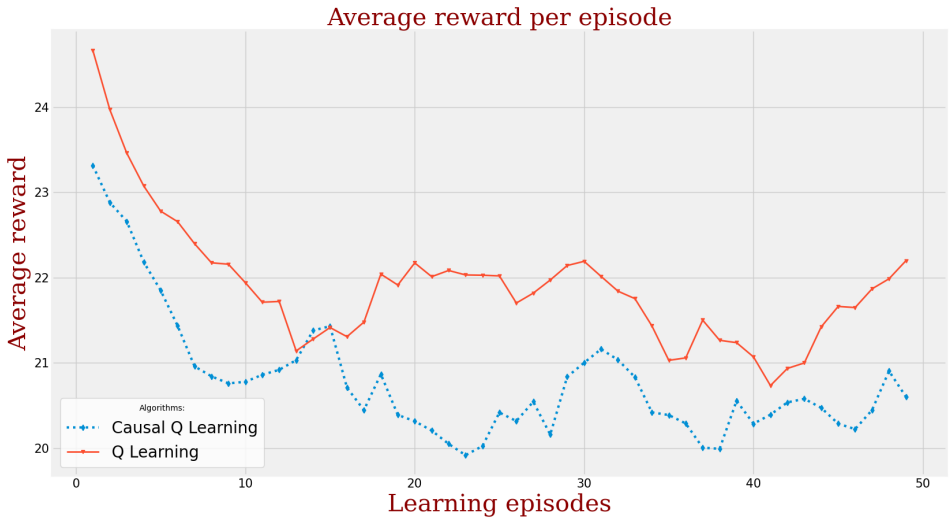


FIGURE 7.19 – Average reward over learning episodes between RL algorithms in MDPUC scenario

result has been drawn from a single Cloud scenario (the one described above) and need to be further investigated for generalisation. However, this is a good beginning to demonstrate the gain in MDPUC scenarios.

7.5 Summary of the chapter

In this chapter we have treated Causal Reinforcement Learning for multi-tier network applications. After presenting the causality domain, we have shown how the CRL emerging field could integrate the counterfactual reasoning in RL methods. Last we have applied the CRL algorithm on the tandem queue environment (with and without hidden variables) and demonstrated the gain of using causal models in RL algorithms. Moreover, it is believed that in many industrial applications, a causal model is already known by human experts and can be implemented in the software agent (robotics, vehicles, networks, etc.). In other fields, where researchers ought to discover causal influences, RL can also be important for causal discovery.

In addition, we have built an SCM-simulator that can express the different MDP scenarios, namely : MDP, POMDP and MDPUC. This simulator acts as a generalisation process that can simulate different environments and may be easier to work with for assessing CRL methods and comparing with SoTA RL techniques. The python simulator takes as input a Directed acyclic graph *DAG* and a dictionary for all nodes of the graph with characteristics such as *observable, noise variable, state variable, action variable, causal assignments, etc.*. An OpenAI Gym simulator with implemented *step(action)* method simulate a one time step trajectory, i.e. going from time t to time $t+1$, and return new observation at time $t+1$ and a reward. It is believed such investigations could highly help to assess new CRL methods and to help the generalisation of the methods.

CHAPTER 8

DISCUSSION

In order to conclude this work, three sections are presented to summarise the contributions of the thesis and to give the resulting perspectives. The first section is a taxonomy of the Reinforcement Learning field that ought to unified all the methods we have seen throughout this document in the different chapters. This allows us to have a broad overview of the field. The second section presents a summary of the contributions of the thesis. It reviews the contributions of each chapter, mentioning the important points. We finish this document with the presentation of the perspectives that derive from the different works. Here again we mention the general perspectives of the thesis but also specific points in each chapter.

8.1 Taxonomy of the Reinforcement Learning field

In this section we want to characterise the different RL methods and their behavior in different environments with what we have seen throughout the manuscript in the different chapters (model-free RL, model-based RL, factored and causal RL). We mainly try to give a global definition to all the possible elements of the RL field : model representation, knowledge of the environment, how the agent can learn the policy or the value function.

General RL Algorithms and Agent's Knowledge

First, we describe the potential knowledge the learning agent can carry and that he can think as a toolbox to learn an optimal policy. Basically, we have two major domains : the MDP domain where the agent knows perfectly the environment model and can plan ; and the RL domain where it has some missing information and needs to interact for data collection.

Agent's knowledge of the environment

First, the learning agent may have disparate understanding of the environment, i.e. different information about the systems statistics but also different representation of the world model.

❒ Markov Decision Process with full information about dynamics and reward functions :

- *State-transitional* model $\mathcal{P}, \mathcal{R}$ [MDP] ;
- *Factored state-transitional* model Fact($\mathcal{P}$), Fact($\mathcal{R}$) with DBNs structure $\mathcal{D}$ [FMDP] ;
- *Structural causal* model $\mathcal{F}, \mathcal{P}(U)$ with causal graph structure $\mathcal{G}$ [CMDP].

❒ Reinforcement Learning with partial (or none) information about dynamics :

— Zero information ;

— Relational structures DBNs or SCM /Causal Graph (and causal assignments) but randomness is unknown (statistics in CPTs are unknown, $P(U)$ is unknown);

Overall we dispose of three main representations to solve complex dynamic problems : Markov Decision Process (MDP) (Chapter IV and Chapter V), Factored Markov Decision Process (FMDP) (Chapter VI) and Causal Markov Decision Process (CMDP) (Chapter VII). The three representations are linked by the following ladder (Figure 8.1) where the most powerful tool is at the top of the ladder. We describe the representational power of the model as the knowledge the agent can carry.

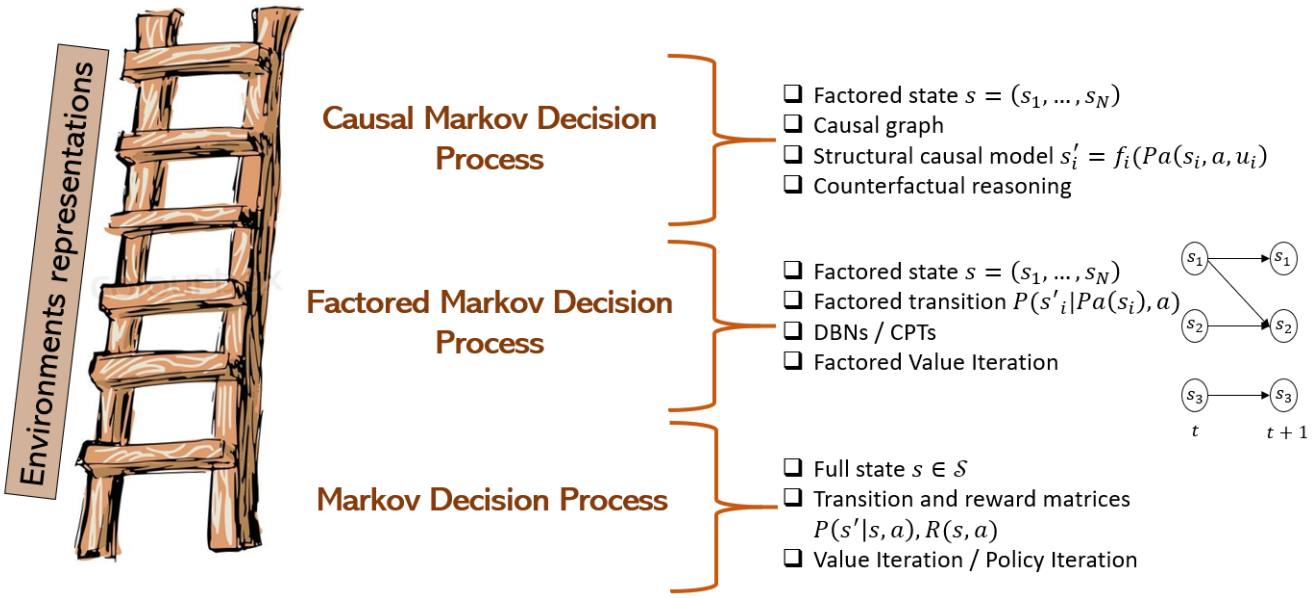


FIGURE 8.1 – Ladder of representation for environments dynamics

How to learn ?

Given the agent's knowledge, it can more or less learn an efficient policy. We describe here several classes of RL algorithms an autonomous agent can use to learn an optimal policy, ranging from model-free to pure model-based methods.

❒ Class 1. The agent has zero information and tries to learn optimal policy without any model, simply by interaction and updates of its policy/value function → Model-free RL;

❒ Class 2. The agent has some information about the relational structure of the world but does not quantify the statistics. In other words, it knows $\mathcal{D}$ but not the CPTs or $\mathcal{G}$ but not the noise distribution $P(U)$. Therefore it interacts with the environment to collect samples and update the statistics. It plans with its updated model to update its policy/value function → Light Model-based;

❒ Class 3. The agent has zero information but wants to learn a model of the world for planning. Therefore it interacts with the environment to collect samples to update the relational structure (DBNs, causal inference (CI) for $\mathcal{G}$) and update the statistics. It plans with its updated model to update its policy/value function → Hard Model-based;

❒ Class 4. The agent has full information and does not need to interact with the environment, it can already calculate offline the optimal policy by planning with dynamic programming techniques → Pure Model-based RL ∼ MDP.

We provide more details about the different model representations that can be used in different RL classes.

Method i (state-transitional model-based RL) treats with $\mathcal{P}$ and $\mathcal{R}$. Consider a discrete state space, the RL agent can learn a tabular model for transitions $\mathcal{P}$ and reward $\mathcal{R}$. If the state space is continuous or discrete but uncountable for tabular implementation, then RL agent has to learn an approximated model of the dynamics $\tilde{\mathcal{P}}$ and reward $\tilde{\mathcal{R}}$. Such model could be represented by functional form such as neural networks, Gaussian process, linear approximation, etc, which makes it an approximation due to function design and updated parameters.

Method ii (factored state-transitional model-based RL) treats with Fact($\mathcal{P}$) and Fact($\mathcal{R}$). Consider a discrete state space, the RL agent can learn a tabular model for CPTs Fact($\mathcal{P}$) and reward Fact($\mathcal{R}$). If the state space is continuous or discrete but uncountable for tabular implementation, then RL agent has to learn an approximated model of the dynamics Fact($\tilde{\mathcal{P}}$) and reward Fact($\tilde{\mathcal{R}}$). Such model could be represented by functional form such as neural networks, Gaussian process, linear approximation, etc, which makes it an approximation due to function design and updated parameters.

Method iii (SCM model-based RL) assume that RL agent fully knows structural causal assignments $\mathcal{F}$ between environment variables but does not know noise distribution $\mathcal{P}(\mathcal{U})$. In this context the agent can use its SCM to perform counterfactual reasoning as described in Section 7.2.5.

We would also like to point out that in all classes and methods, the policies or value functions can be tabular in countable discrete state spaces or can be a functional approximation such as neural networks in uncountable discrete/continuous state spaces.

Summary of RL algorithms

By mixing classes and methods we summarise all the Reinforcement Learning algorithms in Table 8.1.
⊗ refers to "Unknown", ∼ refers to "To learn/update", √ refers to "Known".

Class	*Methods*	*Properties*				
		Relational Knowledge	Model Representation	Relational Structure	Statistics	Name
Class 1.	Method i.	⊗	⊗	⊗	⊗	Model-free algorithms
	Method ii.	⊗	⊗	⊗	⊗	
	Method iii.	⊗	⊗	⊗	⊗	
Class 2.	Method i.	⊗	$\mathcal{P}, \mathcal{R}$	⊗	∼	DYNA / MDP online
	Method ii.	DBNs $\mathcal{D}$	Fact($\mathcal{P}$), Fact($\mathcal{R}$)	√	∼	SDYNA / FMDP Online
	Method iii.	Causal Graph $\mathcal{G}$	$\mathcal{F}, \mathcal{P}(U)$	√	∼	CDYNA
Class 3.	Method i.	⊗	$\mathcal{P}, \mathcal{R}$	⊗	∼	DYNA / MDP online
	Method ii.	DBNs $\mathcal{D}$	Fact($\mathcal{P}$), Fact($\mathcal{R}$)	∼	∼	SDYNA / FMDP Online + DBN learning
	Method iii.	Causal Graph $\mathcal{G}$	$\mathcal{F}, \mathcal{P}(U)$	∼	∼	CDYNA + CI
Class 4.	Method i.	⊗	$\mathcal{P}, \mathcal{R}$	⊗	√	VI, PI
	Method ii.	DBNs $\mathcal{D}$	Fact($\mathcal{P}$), Fact($\mathcal{R}$)	√	√	FVI, FPI, SVI, SPI
	Method iii.	Causal Graph $\mathcal{G}$	$\mathcal{F}, \mathcal{P}(U)$	√	√	CVI (DP with SCM)

TABLE 8.1 – Categorisation of reinforcement learning algorithms

8.2 Conclusion of the thesis

The work presented in this thesis focuses on Reinforcement Learning and optimisation techniques to compute optimal auto-scaling policies in Cloud environments. We have evaluated several RL methods from model-

free to model-based and structural approaches in different Cloud scenarios (single physical node, multi-tier architecture).

The first application (Chapter IV) where we provided the true model to the agent allows us to compare a Markov Decision Process approach against a Markov Chain approach to compute threshold policies for Cloud resource allocation. We provided an aggregation technique to accelerate the computations of the heuristics, based on the Markov Chain aggregation and decomposition of the cost calculations. Moreover, we studied structural hysteresis MDP algorithms with heuristics and we showed that structural MDP algorithms were outperforming heuristics in accuracy and execution time and could handle large-scale systems. These results go against some assumptions in the literature stating that heuristics perform better. Last, we performed analysis in a real Cloud model demonstrating how fast the MDP techniques could compute optimal threshold rules in Cloud systems. These results also encourage Cloud providers to utilise such techniques for thresholds computations after defining consistent queuing metrics for the reward function and after inferring arrivals distributions with traces.

Secondly, we have illustrated in Chapter V that model-based reinforcement learning could outperform state of the art model-free techniques such as Q-Learning in a tandem queue system where the agent has to learn the model of the world from interventional data for planning. This is noteworthy because standard RL applications in industrial use-cases are model-free and we believe that investing a little more in model-based techniques could increase the performance of RL for concrete use-cases.

Again, adding more precise knowledge to the learning agent with structural representation of the environment model (factored and causal), demonstrated that this supplementary information provided to the agent was helping him to obtain better performance. This was shown in the factored approach in Chapter VI (assuming that the environment is consistent with this approach) where more compact representation of the environment model could lead to better performance by highly reducing the complexity of algorithms. We believe that this framework should also be deeper investigated in industrial applications where environments permits its utilisation (end-to-end scenarios with local relations : robotics, logistics, networks, etc.).

In addition, the causal reasoning (Chapter VII) can bring benefits in scenarios with hidden variables which is often the case in practice in industrial applications. Although the field CRL is new and emerging, there is a high interest to investigate in such solutions (theoretically and practically) to democratise CRL in real use-cases. On the other hand, it is believed that causal relationships are intrinsically in nature and that this can be applied in all fields.

8.3 Perspectives of the thesis

This thesis makes the link between model-based reinforcement learning algorithms and the search of auto-scaling policies in Cloud environments. In this section, we provide some directions to deepen our work, in each of the following areas that were cover in the document.

Cloud environments

First, the consideration of more complex Cloud environments is crucial to evaluate the learning of auto-scaling policies. In this document we have investigated two main applications : a single physical node hosting virtual resources and a multi-tier Cloud network with two physical nodes in tandem. To have a deeper evaluation and comparison of RL methods, it is crucial to consider more complex networks; e.g. multi-tier networks with many nodes in tandem or general networks. This could also raise interest to demonstrate the gain of factored and causal approaches in more complex Cloud structures. In addition to increasing the complexity of Cloud infrastructure, it may also be interesting to consider more complex queueing models with general arrivals distributions and services,

with load balancing process, etc. This can be achieved through more complex queueing modelling for Cloud environments representation, for example by looking at the literature or at the behavior of real Cloud platforms.

Simulations and real platform evaluation

Secondly, all our assessments and comparison were made in simulated environments : python simulator for Reinforcement Learning studies and computational software for the treatment of MDP versus heuristics. To give more strength to our work, one major outlook is to evaluate all the investigations in real Cloud platforms. This could be done in experimental Docker platforms developed with Kubernetes or Openstack for example. Moreover, hysteresis and thresholds rules could be evaluated in real Cloud platform such as AwS EC2 or Azure.

Theoretical extensions

Finally, we mentioned throughout this document, mostly during the summary of the chapters, the improvement that could be made regarding the RL algorithms and their comparison. We aggregate here what can be done in each chapter.

Model-based Reinforcement Learning

The work presented in the chapter Chapter V requires deeper comparison and set of parameters to evaluate more globally the considered RL algorithms in Cloud environments. We also want to consider larger-scale scenarios where we might have to approximate policy, value function or models with functional forms such as neural networks, linear approximations, Gaussian processes, etc. Moreover, it could be interesting to investigate the structural form of the optimal policies in the tandem queue environment. This could lead us to integrate this structural properties in the Reinforcement Learning algorithms to speed up the convergence such as in Chapter IV.

Factored Reinforcement Learning

Equally, the factored approach (Chapter VI) should be further evaluated. This should be done by considering larger set of parameters for the simulations, but also more complex Cloud networks. One other path is to compare in very large scale environments the performance of our FMDP online method compared with state of the art Deep RL methods such as DQN or PPO for example. Moreover, we decided in this document to work on tabular representation of the local factored transition probabilities. It can also be of great interest to tackle the problem with decision tree and decision diagram representations.

Causal Reinforcement Learning

For the causal approach many paths can be considered. First on the algorithmic domain there are several elements to investigate regarding the counterfactual reasoning process for different MDP environments (MDPUC, POMDP, etc.). Secondly, there may be an interesting work to be explored regarding the connection between factored and causal approaches, i.e. the relations between DBN and Causal Graph representations. In addition, there is still a lot of evaluation to do for causal methods, in particular with the treatment of partially observable environments with the consideration of historical data.

Hierarchical framework

Finally, one orthogonal approach that was not presented in this document is the *hierarchical framework*. The general idea of *hierarchical MDP* methods is to split the state space into subsets of limited size, compute the local optimal policies for each subset, and then combine these policies to obtain an optimal policy on the on the global MDP. These techniques are related with the aggregation we studied in Chapter IV. The idea is similar in that its goal is to accelerate the convergence by diminishing the dimension of the problem in smaller sub MDPs easier to solve. Moreover, Guestrin et al. studied the intersection between factored and hierarchical MDP with Linear Programming [50]. Kozolava [83] also investigated both approaches and this is one path that could be followed regarding the Chapter VI.

REFERENCES

Book Sources

[1] I. Goodfellow, Y. Bengio et A. Courville. Deep Learning. MIT Press, 2016. URL : http://www.deeplearningbook.org.

[2] G. Pdmia. Markov Decision Processes In Artificial Intelligence. John Wiley & Sons, Inc., 2013. DOI : 10.1002/9781118557426.

[3] J. Pearl. Causality - Models, Reasoning and Inference. 2nd. Cambridge University Press, 2009.

[4] W. B. Powell. Approximate Dynamic Programming. Solving the Curses of Dimensionality. Sous la dir. de J. Wiley et Sons. 2011.

[5] M. Puterman. Markov Decision Processes - Discrete Stochastic Dynamic Programming. Wiley, 1994. DOI : 10.1002/9780470316887.

[6] D.-P. Song. Optimal Control and Optimization of Stochastic Supply Chain Systems. Springer-Verlag, 2013. DOI : 10.1007/978-1-4471-4724-4.

[7] W. Stewart. Introduction to the Numerical Solution of Markov Chains. Princeton, University Press, 1994.

[8] R. Sutton et A. Barto. Reinforcement Learning - An Introduction. The MIT press, 2015.

[9] J. Teghem. Recherche opérationnelle - Tome 1. T. 1. Ellipse, 2013.

Conference and Journal papers

[10] I. Adan, V. Kulkarni et A. van Wijk. « Optimal Control of a Server Farm ». In : Information Systems and Operational Research 51.4 (2013), p. 241–252. DOI : 10.3138/infor.51.4.241.

[11] I. Alam et al. « A Survey of Network Virtualization Techniques for Internet of Things Using SDN and NFV ». In : ACM Comput. Surv. 53.2 (2020). DOI : 10.1145/3379444.

[12] A. Ali-Eldin et al. « Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control ». In : 3rd Workshop on Scientific Cloud Computing (2012).

[13] Amazon. Amazon EC2 Pricing. 2019. URL : https://aws.amazon.com/fr/ec2/pricing/.

[14] Amazon. AWS Auto Scaling. 2018. URL : http://aws.amazon.com/autoscaling/.

[15] Amazon. AWS Serverless Multi-Tier Architectures. AWS Whitepaper. 2019. URL : https://d0.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf.

[16] D. Ardagna et al. « Quality-of-service in cloud computing : modeling techniques and their applications ». In : Journal of Internet Services and Applications 5 (2014). DOI : 10.1186/s13174-014-0011-3.

[17] J. Artalejo, A. Economou et M. Lopez-Herrero. « Analysis of a multiserver queue with setup times ». In : Queueing Systems 51.1-2 (2005), p. 53–76. DOI : 10.1007/s11134-005-1740-6.

[18] N. Asghari, M. Mandjes et A. Walid. « Energy-efficient scheduling in multi-core servers ». In : Computer Networks 59.11 (2014), p. 33–43. DOI : 10.1016/j.bjp.2013.12.009.

[19] D. Bacigalupo et al. « Resource management of enterprise cloud systems using layered queuing and historical performance models ». In : Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing (2010). DOI : 10.1109/IPDPSW.2010.5470782.

[20] E. Bareinboim et al. « On Pearl's Hierarchy and the Foundations of Causal Inference ». In : Technical Report (2021). URL : https://causalai.net/r60.pdf.

[21] E. Barrett, E. Howley et J. Duggan. « A learning architecture for scheduling workflow applications in the cloud ». In : European Conference on Web Services. 2011, p. 83–90. DOI : 10.1109/ECOWS.2011.27.

[22] E. Barrett, E. Howley et J. Duggan. « Applying reinforcement learning towards automating resource allocation and application scalability in the cloud ». In : Concurrency and Computation - Practice and Experience 25.12 (2013), p. 1656–1674. DOI : https://doi.org/10.1002/cpe.2864.

[23] A. G. Barto, S. J. Bradtke et S. Singh. « Learning to Act Using Real-Time Dynamic Programming ». In : Artif. Intell. 72 (1995), p. 81–138.

[24] C. Belady et al. « Green grid data center power efficiency metrics : PUE and DCIE ». In : 2008.

[25] A. Benoit et al. « Reducing the energy consumption of large scale computing systems through combined shutdown policies with multiple constraints ». In : Int. J. High Perform. Comput. Appl. 32.1 (2018), p. 176–188. DOI : 10.1177/1094342017714530.

[26] C. Boutilier, R. Dearden et M. Goldszmidt. « Stochastic dynamic programming with factored representations ». In : Artificial Intelligence 121 (2000), p. 49–107. DOI : 10.1016/S0004-3702(00)00033-3.

[27] R. Brafman et M. Tennenholtz. « R-Max - A General Polynomial Time Algorithm for near-Optimal Reinforcement Learning ». In : J. Mach. Learn. Res. 3 (2003), p. 213–231. DOI : 10.1162/153244303765208377.

[28] M. Braglia et al. « A continuous review, (Q, r) inventory model for a deteriorating item with random demand and positive lead time ». In : Computers and Operations Research 109 (2019), p. 102–121. DOI : 10.1016/j.cor.2019.04.019.

[29] G. Brockman et al. « Openai gym ». In : CoRR abs/1606.01540 (2016). arXiv : 1606.01540.

[30] L. Buesing et al. « Woulda, Coulda, Shoulda : Counterfactually-Guided Policy Search ». In : CoRR abs/1811.06272 (2018). arXiv : 1811.06272.

[31] J. Cao et al. « Web Server Performance Modeling using an M/G/1/K*PS Queue ». In : 10th International Conference on Telecommunications (2003), p. 1501–1506. DOI : 10.1109/ICTEL.2003.1191656.

[32] E. Casalicchio et L. Silvestri. « Autonomic Management of Cloud-Based Systems : The Service Provider Perspective ». In : Springer London, 2013, p. 39–47. DOI : 10.1007/978-1-4471-4594-3_5.

[33] T. Chieu et al. « Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment ». In : nov. 2009, p. 281–286. DOI : 10.1109/ICEBE.2009.45.

[34] C. Colas, O. Sigaud et P.-Y. Oudeyer. A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms. 2019. arXiv : 1904.06979.

[35] T. Degris, O. Sigaud et P.-H. Wuillemin. « Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems ». In : 23rd International Conference on Machine Learning (2006). DOI : 10.1145/1143844.1143877.

[36] M. Deisenroth et C. Rasmussen. « PILCO : A Model-Based and Data-Efficient Approach to Policy Search. » In : jan. 2011, p. 465–472.

[37] G. Ducamp, C. Gonzales et P.-H. Wuillemin. « aGrUM/pyAgrum : a Toolbox to Build Models and Algorithms for Probabilistic Graphical Models in Python ». In : 10th International Conference on Probabilistic Graphical Models. T. 138. Proceedings of Machine Learning Research. Skørping, Denmark, sept. 2020, p. 609–612. URL : https://hal.archives-ouvertes.fr/hal-03135721.

[38] X. Dutreilh et al. « Using Reinforcement Learning for Autonomic Resource Allocation in Clouds : Towards a Fully Automated Workflow ». In : ICAS (2011).

[39] X. Dutreilh et al. « From Data Center Resource Allocation to Control Theory and Back ». In : IEEE 3rd International Conference on Cloud Computing (2010), p. 410–417.

[40] T. Ercan et B. Yusuf. « An Efficient Queuing Model for Resource Sharing in Cloud Computing ». In : The International Journal of Engineering and Science 3 (oct. 2014), p. 36–43.

[41] K. R. Evangelin et V. Vidhya. « Performance Measures of Queuing Models Using Cloud Computing ». In : Asian Journal of Engineering and Applied Technology 4 (2015), p. 8–11.

[42] A. Federgruen et C. So. « Optimality of threshold policies in single-server queueing systems with server vacations ». In : Advances in Applied Probability 23.2 (1991), p. 388–405. DOI : 10.2307/1427755.

[43] W. Fedus et al. « Revisiting Fundamentals of Experience Replay ». In : 2020.

[44] V. Feinberg et al. « Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning ». In : ArXiv abs/1803.00101 (2018).

[45] A. Gandhi, M. Harchol-Balter et I. Adan. « Server farms with setup costs ». In : Performance Evaluation 67.11 (2010), p. 1123–1138. DOI : 10.1016/j.peva.2010.07.004.

[46] Y. Gari et al. « Reinforcement Learning-based Application Autoscaling in the Cloud : A Survey ». In : Engineering Applications of Artificial Intelligence 102 (2021). DOI : 10.1016/j.engappai.2021.104288.

[47] S. Gershman. « Reinforcement learning and causal models ». In : (2015).

[48] H. Ghanbari et al. « Exploring Alternative Approaches to Implement an Elasticity Policy ». In : Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing. CLOUD '11. IEEE Computer Society, 2011, p. 716–723. doi : 10.1109/CLOUD.2011.101.

[49] E. Ghomi, A. Rahmani et N. Qader. « Applying queue theory for modeling of cloud computing : A systematic review ». In : Concurrency and Computation, Practice and Experience 31.17 (2019). doi : https://doi.org/10.1002/cpe.5186.

[50] C. Guestrin, R. Parr et S. Venkataraman. « Efficient solution algorithms for factored MDPs ». In : Journal of Artificial Intelligence Research 19 (2003), p. 399–468.

[51] C. Guestrin, R. Patrascu et D. Schuurmans. « Algorithm-Directed Exploration for Model-Based Reinforcement Learning in Factored MDPs ». In : ICML. 2002.

[52] L. Guo et al. « Dynamic Performance Optimization for Cloud Computing Using M/M/m Queueing System ». In : Journal of applied mathematics (2014). doi : 10.1155/2014/756592.

[53] S. Gupta et A. Sakshi. « Queuing systems in cloud services management : A survey ». In : International Journal of Pure and Applied Mathematics 119 (jan. 2018), p. 12741–12753.

[54] D. Guyon et al. « Involving Users in Energy Conservation : A Case Study in Scientific Clouds ». In : International Journal of Grid and Utility Computing, Inderscience (2018), p. 1–14. doi : 10.1504/IJGUC.2019.10021321.

[55] T. Haarnoja et al. « Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor ». In : 2018.

[56] H. V. Hasselt, M. Hessel et J. Aslanides. « When to use parametric models in reinforcement learning ? » In : NeurIPS. 2019.

[57] M. Hessel et al. « Rainbow : Combining Improvements in Deep Reinforcement Learning ». In : The 32nd AAAI Conference on Artificial Intelligence (2017). url : https://ojs.aaai.org/index.php/AAAI/article/view/11796.

[58] S. Hipp et U. Holzbaur. « Decision Processes with Monotone Hysteretic Policies ». In : Operations Research 36.4 (1988), p. 585–588. url : https://www.jstor.org/stable/171138.

[59] J. Hoey et al. « SPUDD : Stochastic planning using decision diagrams ». In : Uncertainty in Aritificial Intelligence (1999), p. 279–288. url : https://dl.acm.org/doi/10.5555/2073796.2073828.

[60] G. Holland, E. Talvitie et M. Bowling. « The Effect of Planning Shape on Dyna-style Planning in High-dimensional State Spaces ». In : (juin 2018).

[61] S. Horovitz et Y. Arian. « Efficient cloud auto-scaling with SLA Objective Using Q-Learning ». In : IEEE FiCloud. 2018. doi : 10.1109/FiCloud.2018.00020.

[62] E. Hyon et A. Jean-Marie. « Optimal control of admission in service in a queue with impatience and setup costs ». In : Performance Evaluation 144 (2020), p. 102134. issn : 0166-5316. doi : https://doi.org/10.1016/j.peva.2020.102134.

[63] O. Ibe et J. Keilson. « Multi-server threshold queues with hysteresis ». In : Performance Evaluation 21 (1995), p. 185–213. doi : 10.1016/0166-5316(94)E0043-I.

[64] N. Jain et S. Choudhary. « Overview of virtualization in cloud computing ». In : 2016, p. 1–4. doi : 10.1109/CDAN.2016.7570950.

[65] V. Jain, S. Qi et K. Ramakrishnan. « Fast Function Instantiation with Alternate Virtualization Approaches ». In :
2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN). 2021, p. 1–6. doi : 10.1109/LANMAN52105.2021.9478808.

[66] A. Jean-Marie. « marmoteCore : a Markov Modeling Platform ». In : VALUETOOLS. 2017, p. 60–65. doi : 10.1145/3150928.3150960.

[67] L. Jianyong et Z. Xiaobo. « On Average Reward Semi-Markov Decision Processes with a General Multichain Structure ». In : Mathematics of Operations Research 29 (2004), p. 339–352.

[68] Y. Jin et al. « Testing a Q-learning Approach for Derivation of Scaling Policies in Cloud-based Applications ». In : ICIN. 2018. doi : 10.1109/ICIN.2018.8401621.

[69] S. Kakade. « On the Sample Complexity of Reinforcement Learning ». Thèse de doct. 2003.

[70] G. Kalweit et J. Boedecker. « Uncertainty-driven Imagination for Continuous Deep Reinforcement Learning ». In : Proceedings of the 1st Annual Conference on Robot Learning 78 (2017), p. 195–206. url : https://proceedings.mlr.press/v78/kalweit17a.html.

[71] M. M. Kandi et al. « Analysis of Performance and Energy Consumption in the Cloud ». In : Computer Performance Engineering - 14th European Workshop, EPEW. 2017, p. 199–213. doi : 10.1007/978-3-319-66583-2_13.

[72] A. Kansal et al. « Virtual Machine Power Metering and Provisioning ». In :
ACM Symposium on Cloud Computing (2010), p. 39–50.

[73] M. Kearns et D. Koller. « Efficient Reinforcement Learning in Factored MDPs ». In :
Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2. IJCAI'99. Stockholm, Sweden : Morgan Kaufmann Publishers Inc., 1999, p. 740–747. url : https://dl.acm.org/doi/10.5555/1624312.1624325.

[74] M. Kearns et S. Singh. « Near-Optimal Reinforcement Learning in Polynomial Time ». In : Machine Learning 49 (2002), p. 209–232. doi : 10.1023/A:1017984413808.

[75] D. Kenga, V. O. Omwenga et P. J. Ogao. « A Method for Measuring Energy Consumption in IaaS Cloud ». In : JACET (2020).

[76] A. A. Khan, M. Zakarya et R. Khan. « Energy-aware dynamic resource management in elastic cloud datacenters ». In : Simulation Modelling Practice and Theory 92 (2019), p. 82–99.

[77] G. Kibalya et al. « A Reinforcement Learning Based Approach for 5G Network Slicing Across Multiple Domains ». In :
2019 15th International Conference on Network and Service Management (CNSM). 2019, p. 1–5. doi : 10.23919/CNSM46954.2019.9012674.

[78] M. Kitaev et R. Serfozo. « M/M/1 Queues with Switching Costs and Hysteretic Optimal Control ». In : Operations Research 47 (1999), p. 310–312. url : https://www.jstor.org/stable/223048.

[79] D. Koller et R. Parr. « Policy Iteration for Factored MDPs ». In : UAI'00 (2000), p. 326–334. url : https://dl.acm.org/doi/10.5555/2073946.2073985.

[80] J. Koo et al. « Deep Reinforcement Learning for Network Slicing with Heterogeneous Resource Requirements and Time Varying Traffic Dynamics ». In :
2019 15th International Conference on Network and Service Management (CNSM). 2019, p. 1–5. doi : 10.23919/CNSM46954.2019.9012702.

[81] G. Koole. « A simple proof of the optimality of a threshold policy in a two-server queueing system ». In : Systems and Control Letters 26 (1995), p. 301–303. doi : 10.1016/0167-6911(95)00015-1.

[82] P. Koperek et F. Wlodzimierz. « Dynamic Business Metrics-driven Resource Provisioning in Cloud Environments ». In : (2012), p. 171–180. doi : 10.1007/978-3-642-31500-8_18.

[83] O. Kozlova. « Hierarchical and factored reinforcement learning ». Thèse de doct. 2010. url : https://tel.archives-ouvertes.fr/tel-00632968/document.

[84] A. Kranenburg et G. van Houtum. « Cost optimization in the (S-1, S) lost sales inventory model with multiple demand classes ». In : Oper. Res. Lett. 35.4 (2007), p. 493–502. doi : 10.1016/j.orl.2006.04.004.

[85] B. Krishnan et al. « VM Power Metering : Feasibility and Challenges ». In : ACM SIGMETRICS Performance Evaluation Review 38 (2011), p. 56–60.

[86] J. Krzywda et al. « Power-performance tradeoffs in data center servers : DVFS, CPU Pinning, horizontal and vertical scaling ». In : Future Generation Computer Systems (2018), p. 114–128. doi : 10.1016/j.future.2017.10.044.

[87] M. Kurpicz, A.-C. Orgerie et A. Sobe. « How much does a VM cost ? Energy-proportional Accounting in VM-based Environments ». In : Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (2016). doi : 10.1109/PDP.2016.70.

[88] T. Labidi, A. Mtibaa et H. Brabra. « CSLAOnto : A Comprehensive Ontological SLA Model in Cloud Computing ». In : Journal on Data Semantics 5 (2016), p. 179–193. doi : 10.1007/s13740-016-0070-7.

[89] G. V. Lakshmi et C. Bindhu. « A Queuing Model To Improve Quality Service by Reducing Waiting Time in Cloud ». In : International Journal of Soft Computing and Engineering 4 (2014).

[90] N. Lee et V. G. Kulkarni. « Optimal Arrival Rate and Service Rate Control of Multi-server Queues ». In : Queueing Syst. Theory Appl. 76.1 (2014), p. 37–50. doi : 10.1007/s11134-012-9341-7.

[91] Q.-L. Li et al. « An Overview for Markov Decision Processes in Queues and Networks ». In : Stochastic Models in Reliability, Network Security and System Safety (oct. 2019), p. 44–71. doi : 10.1007/978-981-15-0864-6_3.

[92] T. P. Lillicrap et al. « Continuous control with deep reinforcement learning ». In : CoRR abs/1509.02971 (2016).

[93] X. Liu, J. Heo et L. Sha. « Modeling 3-tiered Web applications ». In : (2005), p. 307–310. doi : 10.1109/MASCOTS.2005.40.

[94] Z. Liu, W. Deng et G. Chen. « Analysis of the Optimal Resource Allocation for a Tandem Queueing System ». In : Mathematical Problems in Engineering (2017). doi : 10.1155/2017/5964272.

[95] T. Lorido-Botran, J. Miguel-Alonso et J. Lozano. « A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments ». In : J. Grid Computing 12 (2014), p. 559–592. doi : 10.1007/s10723-014-9314-7.

[96] C. Lu. « Causal Reinforcement Learning : A Road to Artificial General Intelligence ». In : Nokia Bell Labs (2019).

[97] C. Lu, B. Schölkopf et J. M. Hernández-Lobato. « Deconfounding Reinforcement Learning in Observational Settings ». In : ArXiv abs/1812.10576 (2018).

[98] J. C. S. Lui et L. Golubchik. « Stochastic complement analysis of multi-server threshold queues with hysteresis ». In : Performance Evaluation 35 (1 1999), p. 19–48. DOI : 10.1016/S0166-5316(98)00043-1.

[99] V. Maccio et D. Down. « Structural properties and exact analysis of energy-aware multiserver queueing systems with setup times ». In : Performance Evaluation 121-122 (2018), p. 48–66.

[100] J. Magnan. « Représentations graphiques de fonctions et processus décisionnels Markoviens factorisés . » PhD thesis. Université Pierre et Marie Curie - Paris VI, fév. 2016. URL : https://tel.archives-ouvertes.fr/tel-01363858.

[101] M. Mao et M. Humphrey. « A Performance Study on the VM Startup Time in the Cloud ». In : 2012 IEEE Fifth International Conference on Cloud Computing. 2012, p. 423–430. DOI : 10.1109/CLOUD.2012.103.

[102] T. Mastelic et I. Brandic. « Recent Trends in Energy-Efficient Cloud Computing ». In : IEEE Cloud Computing 2 (2015), p. 40–47. DOI : 10.1109/MCC.2015.15.

[103] I. Mitrani. « Managing performance and power consumption in a server farm ». In : Annals of Operations Research 202.1 (2013), p. 121–134. DOI : 10.1007/s10479-011-0932-1.

[104] V. Mnih et al. « Asynchronous Methods for Deep Reinforcement Learning ». In : ICML. 2016.

[105] V. Mnih et al. « Human-level control through deep reinforcement learning ». In : Nature 518 (2015), p. 529–533. DOI : 10.1038/nature14236.

[106] T. Moerland, J. Broekens et C. Jonker. « Model-based Reinforcement Learning : A Survey ». In : CoRR abs/2006.16712v3 (2021). arXiv : 2006.16712v3. URL : http://arxiv.org/abs/2006.16712v3.

[107] R. K. Murugesan, C. Elango et S. Kannan. « Resource Allocation in Cloud Computing with M/G/s- Queueing System ». In : 2014.

[108] A. Nagabandi et al. « Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning ». In : 2018 IEEE International Conference on Robotics and Automation (2018), p. 7559–7566.

[109] V. Ngo. Model-based reinforcement learning. URL : https://ipvs.informatik.uni-stuttgart.de/mlr/wp-content/uploads/2015/06/06-ModelBased.pdf.

[110] J. Niño-Mora. « Resource allocation and routing in parallel multi-server queues with abandonments for cloud profit maximization ». In : Computers and Operations Research 103 (2019), p. 221–236. DOI : 10.1016/j.cor.2018.11.012.

[111] L.-M. L. Ny et B. Tuffin. « A simple analysis of heterogeneous multi-server threshold queues with hysteresis ». In : 2002.

[112] H. Okamura, S. Miyata et T. Dohi. « A Markov Decision Process Approach to Dynamic Power Management in a Cluster System ». In : IEEE Access 3 (2015), p. 3039–3047.

[113] « Optimal resource allocation with deep reinforcement learning and greedy adaptive firefly algorithm in cloud computing ». In : Concurrency and Computation 34 (2022). DOI : 10.1002/cpe.6657.

[114] J. Peters, D. Janzing et B. Schölkopf. « Elements of Causal Inference : Foundations and Learning Algorithms ». In : 2017.

[115] P. Poupart. « Model-based RL, Lesson ». In : University of Waterloo (2018).

[116] C. Qu, R. Calheiros et R. Buyya. « Auto-Scaling Web Applications in Clouds : A Taxonomy and Survey ». In : ACM Comput. Surv. 73.4 (juil. 2018). doi : 10.1145/3148149.

[117] I. Rais et al. « Quantifying the Impact of Shutdown Techniques for Energy-Efficient Data Centers ». In : Concurrency and Computation - Practice and Experience 30.17 (2018), p. 1–13.

[118] A. Randa et al. « Heuristic methods for the capacitated stochastic lot-sizing problem under the static-dynamic uncertainty strategy ». In : Computers and Operations Research 109 (2019), p. 89–101. doi : 10.1016/j.cor.2019.03.007.

[119] J. Rao et al. « VCONF - A Reinforcement Learning Approach to Virtual Machines Auto-Configuration ». In : ICAC '09 (2009), p. 137–146. doi : 10.1145/1555228.1555263.

[120] T. Salah et al. « Performance comparison between container-based and VM-based services ». In : 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN). 2017, p. 185–190. doi : 10.1109/ICIN.2017.7899408.

[121] S. Saxena, M. Khan et R. Singh. « An Energy Saving Approach from Cloud Resources for Green Cloud Environment ». In : 2021 10th International Conference on System Modeling Advancement in Research Trends (SMART). 2021, p. 628–632. doi : 10.1109/SMART52563.2021.9676249.

[122] T. Schaul et al. Prioritized Experience Replay. 2016. arXiv : 1511.05952v4.

[123] J. Schulman et al. « Proximal Policy Optimization Algorithms ». In : ArXiv abs/1707.06347 (2017).

[124] V. Sciancalepore, X. Costa-Perez et A. Banchs. « RL-NSB : Reinforcement Learning-Based 5G Network Slice Broker ». In : IEEE Transactions on Networking 27.4 (2019), p. 1543–1557. doi : 10.1109/TNET.2019.2924471.

[125] S. Scott et P. Smyth. « The Markov Modulated Poisson Process and Markov Poisson Cascade with Applications to Web Traffic Modeling ». In : Bayesian Statistics 7 (2003).

[126] D. Serrano et al. « SLA guarantees for cloud services ». In : Future Generation Computer Systems 54 (2016), p. 233–246. doi : https://doi.org/10.1016/j.future.2015.03.018.

[127] S. Shorgin et al. « Threshold-based queuing system for performance analysis of cloud computing system with dynamic scaling ». In : AIP Conference Proceedings 1648.1 (2015). doi : 10.1063/1.4912509.

[128] G. Siddesh et K. Srinivasa. « SLA - Driven Dynamic Resource Allocation on Clouds ». In : Advanced Computing, Networking and Security. Springer Berlin Heidelberg, 2012, p. 9–18.

[129] D. Silver et al. « A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play ». In : Science 362.6419 (2018), p. 1140–1144. doi : 10.1126/science.aar6404.

[130] P. Singh et al. « Research on Auto-Scaling of Web Applications in Cloud : Survey, Trends and Future Directions ». In : Scalable Computing Practice and Experience 20 (2019). doi : 10.12694/scpe.v20i2.1537.

[131] A. L. Strehl. « Model-Based Reinforcement Learning in Factored-State MDPs ». In : 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. 2007, p. 103–110. doi : 10.1109/ADPRL.2007.368176.

[132] A. L. Strehl, C. Diuk et M. L. Littman. « Efficient Structure Learning in Factored-State MDPs ». In : (2007).

[133] R. S. Sutton. « Dyna, an integrated architecture for learning, planning, and reacting ». In : SIGART Bull 2 (1991), p. 160–163.

[134] D. Szarkowicz et T. Knowles. « Optimal Control of an M/M/S Queueing System ». In : Operations Research 33.3 (1985), p. 644–660.

[135] I. Szita et A. Lörincz. « Factored Value Iteration Converges ». In : Acta Cybernetica 18 (2008), p. 615–635. url : https://dl.acm.org/doi/10.5555/1515867.1515873.

[136] I. Szita et C. Szepesvári. « Model-based reinforcement learning with nearly tight exploration complexity bounds ». In : ICML. 2010, p. 1031–1038. url : https://icml.cc/Conferences/2010/papers/546.pdf.

[137] J. Teghem. « Control of the service process in a queueing system ». In : EJOR 23.2 (1986), p. 141–158. doi : 10.1016/0377-2217(86)90234-1.

[138] G. Tesauro et al. « A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation ». In : Proceedings of the 2006 IEEE International Conference on Autonomic Computing. ICAC '06. IEEE Computer Society, 2006, p. 65–73. doi : 10.1109/ICAC.2006.1662383.

[139] T. Tournaire, H. Castel-Taleb et E. Hyon. « Optimal control policies for resource allocation in the Cloud : comparison between Markov decision process and heuristic approaches ». In : CoRR abs/2104.14879 (2021). url : https://arxiv.org/abs/2104.14879.

[140] T. Tournaire et al. « Factored Reinforcement Learning for Auto-scaling in Tandem Queues ». In : NOMS 2022 - Full and short papers (). Avr. 2022. url : http://XXXXX/220997.pdf.

[141] T. Tournaire et al. « Generating optimal thresholds in a hysteresis queue : a cloud application ». In : IEEE Mascots. 2019. doi : 10.1109/MASCOTS.2019.00040.

[142] T. Tournaire et al. « Reinforcement Learning with Model-Based Approaches for Dynamic Resource Allocation in a Tandem Queue ». In : (2021), p. 243–263. doi : 10.1007/978-3-030-91825-5_15.

[143] B. Urgaonkar et al. « An Analytical Model for Multi-Tier Internet Services and Its Applications ». In : ACM SIGMETRICS Performance Evaluation Review (2005). doi : 10.1145/1064212.1064252.

[144] B. Urgaonkar et al. « Dynamic Provisioning of Multi-Tier Internet Applications ». In : Second International Conference on Autonomic Computing (2005), p. 217–228. doi : 10.1109/ICAC.2005.27.

[145] J. Wang, A. Hertzmann et D. Fleet. « Gaussian process dynamical models ». In : Advances in neural information processing systems (2006), p. 1441–1448.

[146] D. Warsing, W. Wangwatcharakul et R. King. « Computing optimal base-stock levels for an inventory system with imperfect supply ». In : Computers and Operations Research 40 (2013), p. 2786–2800. doi : 10.1016/j.cor.2013.04.001.

[147] C. Watkins et P. Dayan. « Q-Learning ». In : Machine Learning 8 (1992), p. 279–292. doi : 10.1007/BF00992698.

[148] Y. Wei et al. « A Reinforcement Learning Based Auto-Scaling Approach for SaaS Providers in Dynamic Cloud Environment ». In : Mathematical Problems in Engineering 2019 (fév. 2019), p. 1–11. doi : 10.1155/2019/5080647.

[149] C.-H. Wu et al. « Optimization analysis of an unreliable multi-server queue with a controllable repair policy ». In : Computers and Operations Research 49 (2014), p. 83–96. doi : 10.1016/j.cor.2014.03.018.

[150] R. Yang, S. Bhulai et R. van der Mei. « Structural properties of the optimal resource allocation policy for single-queue systems ». In : Annals of Operations Research 202 (2013), p. 211–233. doi : 10.1007/s10479-011-0933-0.

[151] Z. Yang et al. « An Optimal Hysteretic Control Policy for Energy Saving in Cloud Computing ». In : GLOBECOM. 2011, p. 1–5. doi : 10.1109/GLOCOM.2011.6133628.

[152] Y. Yu et S. N. Bhatti. « Energy Measurement for the Cloud ». In : International Symposium on Parallel and Distributed Processing with Applications (2010), p. 619–624.

[153] J. Zhang et E. Bareinboim. « Markov Decision Processes with Unobserved Confounders. A Causal Approach ». In : Report (2016). url : https://www.cs.purdue.edu/homes/eb/mdp-causal.pdf.

[154] J. Zhang, D. Kumor et E. Bareinboim. « Causal Imitation Learning With Unobserved Confounders ». In : NeurIPS. 2020.

[155] Z. Zhou, J. Abawajy et F. Li. « Analysis of Energy Consumption Model in Cloud Computing Environments ». In : (2019), p. 195–215. doi : 10.1007/978-3-319-69889-2_10.

ANNEXE A

APPENDIX

A.1 Stationary distributions of classical queues

As this chapter treats about Markov Chains and Queuing systems, we briefly recall basics about these two frameworks that will be used along this chapter. Markov Decision Process has already been described in Chapter III.

A.1.0.1 Closed forms

The cost function $\tilde{C}_k(m)$ for the coupling of local search heuristics and initialisation is approximated with an approximation of the stationary distribution of the system. Here are presented all the formulas for stationary distribution in queues, that could be useful for the algorithm.

For an M/M/k/∞ queue

$$\pi_k(m) = \begin{cases} \pi_k(0).\dfrac{\rho^m}{m!} & \text{for } 0 \leq m \leq k \\ \pi_k(0).\dfrac{\rho^m}{K^{m-k}.k!} & \text{for } k \leq m \leq B. \end{cases}$$

and with

$$\pi_k(0) = \left(\sum_{m=0}^{k-1} \frac{\rho^m}{m!} + \sum_{m=k}^{B} \frac{\rho^m}{k^{m-k}.k!} \right)^{-1}$$

For an M/M/k/B queue

$$\pi_k(m) = \begin{cases} \pi_k(0).\dfrac{\rho^m}{m!} & \text{for } 1 \leq m \leq k \\ \pi_k(0).\dfrac{a^m.k^k}{k!} & \text{for } k < m \leq B. \end{cases}$$

where $\rho = \dfrac{\lambda}{\mu}$ and $a = \dfrac{\rho}{k} < 1$ and where $\pi_k(0) = \left(\sum\limits_{m=0}^{k-1} \dfrac{\rho^m}{m!} + \dfrac{\rho^k}{k!} \dfrac{a}{1-a} \right)^{-1}$.

We can also write $\pi_k(0)$ as :

$$\pi_k(0) = \begin{cases} \left(\sum_{m=0}^{k-1} \frac{\rho^m}{m!} + \frac{\rho^k}{k!}(B-k+1) \right)^{-1} & \text{if } \rho = 1, \\ \left(\sum_{m=0}^{k-1} \frac{\rho^m}{m!} + \frac{\rho^k}{k!} \frac{1-a^{B-k+1}}{1-a} \right)^{-1} & \text{if } \rho \neq 1. \end{cases}$$

For an MM1B with rate $k.\mu$ Let $a = \dfrac{\lambda}{k * \mu}$ depending on which level we are.
We have :

$$\forall m = 0, \cdots, B, \pi_k(m) = \begin{cases} \frac{1}{B+1} & \text{if } \rho = 1, \\ \frac{(1-a)a^m}{1-a^{B+1}} & \text{if } \rho \neq 1. \end{cases}$$

A.1.1 Computation of the stationary distribution

In Chapter IV we refer to two algorithms for computations of the Markov chain stationary distribution in the general case and also in the aggregation settings for micro-chains : *GTH* and *Power method* [7],[66].

A.1.1.1 GTH algorithm

Algorithm 32: Markov chain resolution with GTH

Input: Generator matrix Q
Output: π

Consider the transposed of the infinitesimal generator $\mathcal{Q}^T$
for $i = 1, 2, \cdots, n-1$ **do**
  Compute $\mu_i = q(i+1, i) + q(i+2, i) + \cdots + q(n, i)$
  for $j = i+1, i+2, \cdots, n$ **do**
    Compute $q(i, j) = q(i, j)/\mu_i$
    for $k = i+1, i+2, \cdots, n$ avec $k \neq j$ **do**
      $q(j, k) = q(j, k) + q(j, i) * q(i, k)$
We obtain the final matrix $\mathcal{Q}'$
Let $x_n = 1$
for $i = n-1, \cdots, 1$ **do**
  $x_i = \sum_{j=i+1}^{n} q(i, j) * x_j$
/* Normalisation */
Let $Norm = \sum_{i=1}^{n} x_i$
for $i = 1, \cdots, n$ **do**
  $\pi_i = x_i / Norm$

A.1.1.2 Power method

The Power method algorithm requires to uniformise the continuous time Markov chain first to solve $\pi\mathcal{P} = \mathcal{P}$.

Algorithm 33: The power method

Input: Probability transition matrix P, K periodical renormalisations, ϵ precision
Output: π

```
/* Begin                                                    */
```

$x \leftarrow$ any probability distribution
$n \leftarrow 0$
repeat
  $n \leftarrow n + 1$
  $y \leftarrow x \times P$
  $dist \leftarrow ||x - y||$
  $x \leftarrow y$
  if $nmodK = 0$ **then**
    $x \leftarrow$ Renormalise(x)
until $dist < \epsilon$
$\pi \leftarrow$ Renormalise(x)
return π

Algorithm 34: Renormalise(x) : Renormalisation of a vector

Input: A vector of numbers x indexed by $\sharp x$
Output: A vector proportional to x which entries sum up to 1

Let $S \leftarrow 0$
for $i \in \sharp x$ **do**
  $S \leftarrow S + x[i]$
for $i \in \sharp x$ **do**
  $x[i] \leftarrow x[i]/S$
return x

RESUME

11 rue Silly
Boulogne-Billancourt 92100
France
tournaire.thomas@hotmail.fr

French
July 17th, 1993
+6 35 34 72 57
thomas.tournaire@nokia.com

Research interest and skills

— Reinforcement learning (RL) methods : performance enhancements with structured model-based approaches (factored, causal, hierarchical)
— RL-based solutions for complex network optimization (Multi-Queuing systems and 5G use-cases)
— Programming skills : Python (Keras, Tensorflow, Numpy, Pandas, Scipy, Simpy), MatLab-Simulink, C++

Work Experience

2019-2022	Research Engineer at Nokia Bell Labs France, Nozay AI research, Maths&Algorithms Root Cause Analysis
2021-2021	Contract Teacher at University Nanterre Paris X Machine Learning, Neural Networks and Deep RL course, M2 D3S-IES Python-VBA course, M2 ISEFAR
2018-2019	Research Intern at Telecom SudParis, Evry Stochastic optimisation of performances and energy consumption in the cloud, PGMO-BOBIBEE Stochastic optimisation problem addressed using combinatorial optimisation methods, Markov Chain, performance evaluation and stochastic dynamic programming methods

Education

2019–2022	**PhD** in Computer Science, IP Paris, Telecom sudParis Title : Reducing energy consumption in largescale networks with MDP and RL Supervisors : Hind Castel-Taleb, Emmanuel Hyon, Armen Aghasaryan
2017–2018	**Master 2** Mathematical Modelling and Methods in Economy and Finance Optimisation and Operational Research track, mention Bien University Paris 1, Panthéon-Sorbonne, France
2016-2017	**Master 1** International Master in Mathematics applied to Economics and Finance, mention Bien, 1st Master Thesis : Measure Theory, Extension of Measures by Kolmogorov and Lebesgue and mathematical Expectation Advisor : Bernard De Meyer
2013-2016	**Bachelor** Mathematics and Informatics applied to human and social sciences, mention Très Bien, 1st University Versailles Saint-Quentin en Yvelines
2011-2012	**Baccalauréat** série S, Spécialité Mathématiques, Mention Très Bien Lycée La Bruyère, Versailles

Prizes

* Winner of the Nokia French Student Award, Bell Labs Section, July 2020

Publications

In preparation

— Optimal control policies for resource allocation in the Cloud : comparison between Markov decision process and heuristic approaches, T. Tournaire, H. Castel-Taleb, E. Hyon, submitted to the journal *Simulation Modelling Practice and Theory* and available at *arXiv preprint arXiv :2104.14879*

International conferences with proceedings and referees

[1] Factored reinforcement learning for auto-scaling in tandem queues, *IEEE NOMS*, Apr. 2022, T. Tournaire, Y. Jin, A. Aghasaryan, H. Castel-Taleb, E. Hyon

[2] Reinforcement learning with model-based approaches for dynamic resource allocation in a tandem queue, *26th International Conference on Analytical & Stochastic Modelling Techniques & Applications*, Aug. 2021, T. Tournaire, J. Barthélémy, H. Castel-Taleb, E. Hyon

[3] Generating optimal thresholds in a hysteresis queue : application to a cloud model, *IEEE Mascots*, p. 283-294, Oct. 2019, T. Tournaire, H. Castel-Taleb, E. Hyon, T. Hoche (24% selection rate)

National conferences with proceedings and referees

[1] Génération de seuils optimaux dans une file hystérésis, *CORES*, Jun. 2019, T. Tournaire, H. Castel-Taleb, E. Hyon, T. Hoche (50% selection rate)

National conferences with abstract selection

* Factored model-based RL for tandem queues resource allocation, *PGMO*, Dec. 2021, T. Tournaire, Y. Jin, A. Aghasaryan
* Comparaisons de méthodes de calcul de seuils pour minimiser la consommation énergétique d'un cloud, *ROADEF*, Nov. 2019, T. Tournaire, H. Castel-Taleb, E. Hyon
* Stochastic optimisation of energy consumption and performance in a cloud system, *PGMO*, Dec. 2018, T. Tournaire, H. Castel-Taleb, E. Hyon

Patents

[P1] AI/ML based PDCP Split in Dual/Multi Connectivity Configurations : Collective Inference/Decision and Impact on 3gpp Interfaces, F. Syed Muhammad, R. Damodaran, T. Tournaire, A. Feki, L. Maggi, F. Durand, U. Chowta (submitted March 29, 2021)

[P2] Multi-Agent Reinforcement Learning Sharing Common Neural Network Policy for PDCP Data Split, T. Tournaire, F. Syed Muhammad, A. Feki, L. Maggi, F. Durand (filed November 11, 2021)

Diverse presentations

* Causal RL, *Nokia Bell Labs Show & Tell*, Mar. 2021
* Causal RL, *Nokia France PhD Seminar*, Mar. 2021
* SimPy package tutorial, *LINCS*, Nov. 2020
* Model-Based RL, *RL Reading Group in Hyper Industrial Analytics Group* at Nokia Bell Labs, Mar. 2020
* Performance and Cost Optimisation in the future 5G Cloud, Poster, *Doctorant Day TSP*, Dec. 2019

Teaching

* VBA/Python TD, M2 ISEFAR, University Paris Nanterre, 2020 (12h), 2021 (22h)
* ML, Neural Networks, Deep RL course, M2 D3S-IES, 2021 (8h)

Intern supervision

* François Pic : Hierarchical MDP for tandem queue, with H. Castel and E. Hyon, 2021
* Jeanne Barthélémy : Model-based RL for tandem queue, with H. Castel and E. Hyon, 2020

Trainings

* Ethics in Scientific Research, February 2022, IP Paris
* Reinforcement Learning virtual school, ANITI, Mar. 2021
* Cosmos : Optimisation and Reinforcement Learning, Nov. 2019, INRIA, Paris
* ResCom summer school : models and methods of network analysis, June 2019, Anglet

* Data Science Working Group
  — From machine learning basics to Feed Forward Neural Networks, 2019, Telecom SudParis
  — Optimisation for machine/deep learning, 2019, Telecom SudParis
  — Queuing theory seminar : A. Brandwajn Talk, 2019, Telecom SudParis
* Student Volunteer at IEEE Infocom, May 2019, Paris
* AI ENS workshop : Machine Learning and User Decision Making, May 2019, Paris
* Nokia Bell Labs Trainings : Amazon Web Services day, Oct. 2019 and World Class Presentation, Dec. 2019, Nozay

Languages

French	Mother tongue
English	Fluent

RÉSUMÉ EN FRANÇAIS ÉLARGI

Introduction

L'émergence de nouvelles technologies (villes intelligentes, milieu hospitalier, véhicules autonomes, IoT, etc.) nécessite une allocation efficace des ressources pour satisfaire la demande. Cependant, ces nouveaux besoins nécessitent une puissance de calcul élevée impliquant une plus grande consommation d'énergie notamment dans les infrastructures cloud et data centers. Il est donc essentiel de trouver de nouvelles solutions qui peuvent satisfaire ces besoins tout en réduisant la consommation d'énergie des ressources. Dans cette thèse, nous proposons et comparons de nouvelles solutions d'IA (apprentissage par renforcement RL) pour orchestrer les ressources virtuelles dans les environnements de réseaux virtuels de manière à garantir les performances et minimiser les coûts opérationnels. Nous considérons les systèmes de file d'attente comme un modèle pour les infrastructures cloud IaaS (Infrastructure as a Service) et apportons des méthodes d'apprentissage pour allouer efficacement le bon nombre de ressources. Notre objectif est de minimiser une fonction de coût en tenant compte des coûts de performance et opérationnels/énergétiques. Nous utilisons différents types d'algorithmes de RL (du « sans-modèle » au « modèle relationnel ») pour apprendre la meilleure politique. L'apprentissage par renforcement s'intéresse à la manière dont un agent doit agir dans un environnement pour maximiser une récompense cumulative. Nous développons d'abord un modèle de files d'attente d'un système cloud avec un nœud physique hébergeant plusieurs ressources virtuelles. Dans cette première partie, nous supposons que l'agent connaît le modèle (dynamiques de l'environnement et coût), ce qui lui donne la possibilité d'utiliser des méthodes de programmation dynamique pour le calcul de la politique optimale. Puisque le modèle est connu dans cette partie, nous nous concentrons également sur les propriétés des politiques optimales, qui sont des règles basées sur les seuils et l'hystérésis. Cela nous permet d'intégrer la propriété structurelle des politiques dans les algorithmes MDP. Après avoir fourni un modèle de cloud concret avec des arrivées exponentielles avec des intensités réelles et des données d'énergie pour le fournisseur de cloud, nous comparons dans cette première approche l'efficacité et le temps de calcul des algorithmes MDP par rapport aux heuristiques construites sur les distributions stationnaires de la chaîne de Markov des files d'attente.
L'objectif affiché de la thèse est double. Il s'agit d'une part d'apprendre la politique optimale d'allocation de ressources dans un environnement de type cloud et d'autre part de concevoir des méthodes d'apprentissages qui soient à la fois efficientes et adaptées aux problématiques auxquelles est confronté le monde industriel. Les contributions de la thèse sont réparties en plusieurs chapitres, résumées ci-dessous.

Littérature

Ce chapitre présente une revue de la littérature sur le Cloud computing, la gestion de l'énergie, les modèles de file d'attente et les techniques d'optimisation pour l'allocation de ressources dans les systèmes à grande échelles (centres de calcul, centres de stockages, etc.). Il présente tout d'abord ce qu'est le Cloud, comment il se comporte, les infrastructures et modèles existants (par exemple, les centres de données) et le cadre de la virtualisation. Ensuite, il donne un aperçu du domaine de la gestion de l'énergie en présentant le besoin, les solutions pour mesurer et modéliser l'énergie dans le Cloud ainsi que le paradigme d'allocation dynamique des ressources. Enfin, il présente les politiques d'auto-dimensionnement dans le Cloud existantes avec des règles basées sur des seuils, des modèles de file d'attente et des techniques de gestion du contrôle et d'apprentissage par renforcement. Pour optimiser la consommation énergétique des systèmes considérés, la méthode retenue dans ce document est la mise en veille des ressources inutilisées. Il s'agit donc d'activer et de désactiver des noeuds de calcul au cours du temps selon la demande et les engagements pris sur le niveau de service.

Apprentissage par renforcement

Le chapitre 3 présente plus en détail les méthodes d'apprentissage qui seront considérées et étudiées dans les chapitres suivants. On introduit les bases des processus de décision Markoviens avec leurs différentes méthodes de résolution et les bases de l'apprentissage par renforcement. Les méthodes sont présentées selon le degré de connaissance du contrôleur, allant de la connaissance parfaite du modèle du système (dynamique et coût) à l'absence de modèle. La présentation ici préfigure l'ordre de présentation des contributions dans les chapitres suivants. Ce chapitre fournit également un bref résumé de deux cas d'applications industrielles de l'apprentissage renforcé sans modèle. Le retour d'expérience apparait mitigé puisque les méthodes utilisées se sont révélées intéressantes et compétitives dans un cas sur les deux. Le chapitre se conclut sur une comparaison des avantages et des inconvénients des méthodes avec ou sans modèle.

Processus de décision Markovien ou Heuristiques : Comment calculer les politiques d'auto-dimensionnement dans un système de files d'attente avec un modèle connu

Dans ce chapitre, nous considérons que l'agent reçoit le véritable modèle de l'environnement $\mathcal{M}$ et nous situons donc dans la classe des algorithmes d'apprentissage par renforcement avec modèle, où le modèle est préalablement connu par l'agent. Nous considérons un problème d'optimisation d'un coût global tenant compte de différents coûts : associés aux exigences de performance définies dans un accord de niveau de service (SLA) et des coûts représentant la consommation d'énergie. Nous voulons calculer la politique optimale d'activation et de désactivation des serveurs en fonction de l'occupation des requêtes dans la file d'attente. Dans ce chapitre, nous étudions des modèles dans lesquels la structure de la politique est de type hystérésis. Cependant, supposer que la politique a une forme d'hystérésis n'est pas suffisant pour caractériser complètement la politique optimale. Il faut déterminer les valeurs concrètes des seuils. A cette fin, il existe deux approches de modélisation qui correspondent à deux courants majeurs de la littérature. La première exprime le problème dans un modèle de problème d'optimisation déterministe. Elle est réalisée par l'utilisation de la valeur moyenne des coûts calculée par la distribution stationnaire d'une CTMC (Continuous Time Markov Chains). La seconde exprime le problème sous un modèle MDP avec des politiques structurées et calcule la politique optimale. Mais, bien qu'elles soient les plus utilisées pour le calcul des seuils, l'efficacité de ces deux approches

n'est pas évaluée très soigneusement pour le calcul des seuils uniques par niveau. En particulier, cela n'a jamais été fait pour les politiques d'hystérésis. Peu de travaux se sont concentrés sur l'optimalité de la politique d'hystérésis pour les modèles de Cloud mais, à notre connaissance, aucun sur le calcul des seuils. Déterminer quelle est la meilleure approche prometteuse et quel est le meilleur modèle à choisir dans une utilisation pratique est donc un point important qui est abordé ici.
Nous validons également notre approche avec un modèle réel. L'approche est difficile car, à notre connaissance, il n'existe pas, dans la littérature, un modèle unifié incluant à la fois l'énergie, la qualité de de service et les paramètres du trafic réel. Nous proposons de construire un coût global prenant en compte à la fois la consommation réelle d'énergie, les coûts financiers réels des machines virtuelles et de l'accord de niveau de service (SLA) simultanément. Nous proposons de construire un coût global prenant en compte à la fois la consommation d'énergie réelle, les coûts financiers réels des machines virtuelles et les accords de niveau de service (SLA). Nous pensons que le modèle présenté est suffisamment générique pour représenter une large classe de problèmes et est suffisant pour donner une une signification pertinente à tous les paramètres. Grâce à nos algorithmes et à ce modèle, les gérants de Cloud peuvent générer des coûts optimisés significatifs dans des cas réels.
Toutefois, ce chapitre émet une hypothèse très forte : assummer que l'agent connait le modèle exact de l'environnement. En pratique, c'est rarement le cas et il faut trouver des solutions plus adaptées qui peuvent apprendre une politique optimale dans des environnements partiellement connus.

Apprentissage par renforcement basé sur un modèle pour les réseaux multi-tiers

Un des défi majeur de l'allocation des ressources dans le Cloud porte sur le fait que les experts ne connaissent pas facilement les statistiques exactes du système (arrivées des requêtes, taux de services) dans les environnements réels, car ils évoluent trop rapidement. Par conséquent, un agent logiciel ne possède aucun modèle précis de l'environnement pour le calcul de la politique optimale. Cela nécessite l'implémentation de solutions d'apprentissage par renforcement qui permettent l'apprentissage de politiques dans des environnements inconnus. D'autre part, nous ne sommes pas sûrs que les approches des praticiens soient robustes (souvent très expérimentales). La plupart des travaux concernant l'apprentissage par renforcement pour l'allocation de ressources dans le Cloud sont consacrés à des techniques sans modèle telles que l'algorithme du Q-Learning ou ou de l'apprentissage par renforcement profond, mais il n'existe aucune application de techniques basées sur des modèles pour de tels problématiques.
Dans ce chapitre, nous voulons évaluer les approches d'apprentissage par renforcement basées sur les modèles dans le cas des réseaux de files d'attente, ce qui a été peu fait jusqu'à présent. De plus, nous voulons étendre la gamme des modèles étudiés dans le chapitre précédent pour aller au-delà du simple modèle multi-serveur en considérant une application réseau avec plusieurs files d'attente, ce qui permet de mieux représenter les architectures multi-tiers, ce qui n'a pas non plus été beaucoup fait dans la littérature. Par conséquent, nous supposons maintenant que l'agent logiciel ne porte pas la connaissance du modèle MDP. L'objectif est donc d'adopter les approches RL basées sur les modèles, en particulier les architectures Dyna et les techniques de controle adaptatif afin d'évaluer leur pertinence, si jamais, ou de les améliorer pour l'allocation dynamique des ressources dans les systèmes de réseaux de files d'attente. De plus, nous voulons évaluer la robustesse de ces approches dans des cas d'utilisation où les arrivées peuvent arriver massivement de manière très brusque.
Ce chapitre donne d'abord un aperçu plus détaillé des techniques d'apprentissage par renforcement basées sur des modèles, puis présente deux modèles de file d'attente pour exprimer les architectures de réseau multi-tiers. Il montre ensuite comment nous avons sélectionné les techniques d'apprentissage par renforcement basées sur des

modèles en fournissant un algorithme général et comment nous les avons paramétrées pour les comparer à l'état de l'art de l'apprentissage par renforcement sans modèle sur un réseau Cloud multi-tier. Enfin, nous présentons les résultats de la comparaison entre les techniques RL sur différents scénarios et paramètres de Cloud dans un environnement simulé, démontrant le gain des techniques RL basées sur le modèle.

Apprentissage par renforcement factorisé pour les réseaux multi-tiers

Dans les chapitres précédents, nous n'avons considéré que des représentations tabulaires de la dynamique et des modèles de récompense, ce qui peut s'avérer peu pratique dans des environnements à grande échelle avec des espaces d'état et d'action de très grande taille. Le cadre de l'apprentissage par renforcement factorisé permet de représenter de manière plus compacte le modèle $\mathcal{M}$ de l'environnement en intégrant les relations entre les variables d'environnement. Dans les environnements qui ont des relations spécifiques entre les variables d'état, il pourrait être très bénéfique d'utiliser cette représentation pour accélérer la convergence et la facilité d'implémentation.
La dernière partie, divisé en deux chapitres, se concentre donc sur les techniques de RL basées sur des modèles avec une structure relationnelle entre les variables d'état. Comme ces réseaux en tandem ont des propriétés structurelles dues à la forme de l'infrastructure Cloud, nous intégrons les approches factorisées et causales aux méthodes de RL pour inclure cette connaissance. Nous fournissons à l'agent une connaissance relationnelle de l'environnement qui lui permet de comprendre comment les variables sont reliées. L'objectif principal est d'accélérer la convergence : d'abord avec une représentation plus compacte avec la factorisation où nous concevons un algorithme en ligne de MDP factorisé que nous comparons avec des algorithmes de RL sans modèle et basés sur un modèle; ensuite en intégrant le raisonnement causal et contrefactuel qui peut traiter les environnements avec des observations partielles et des facteurs de confusion non observés.

Conclusion

Le chapitre 8 présente une nouvelle taxonomie de toutes les méthodes d'apprentissage abordées et mentionnées dans la thèse, avec une vision structurelle de l'apprentissage par renforcement découplé en 3 socles : MDP, MDP factorisé et MDP causal. Finalement, le chapitre reprend toutes les contributions et leurs perspectives, et notamment les différentes pistes envisagées afin de consolider la partie expérimentale.

DOCTORAL SCHOOL

Titre : Apprentissage par renforcement basé sur un modèle pour l'allocation dynamique des ressources dans les environnements Cloud

Mots clés : Apprentissage par renforcement, Politiques d'auto-scaling, Cloud, Hysteresis, Apprentissage par renforcement factorisé, Apprentissage par renforcement causal

Résumé : L'émergence de nouvelles technologies nécessite une allocation efficace des ressources pour satisfaire la demande. Cependant, ces nouveaux besoins nécessitent une puissance de calcul élevée impliquant une plus grande consommation d'énergie notamment dans les infrastructures cloud et data centers. Il est donc essentiel de trouver de nouvelles solutions qui peuvent satisfaire ces besoins tout en réduisant la consommation d'énergie des ressources. Dans cette thèse, nous proposons et comparons de nouvelles solutions d'IA (apprentissage par renforcement RL) pour orchestrer les ressources virtuelles dans les environnements de réseaux virtuels de manière à garantir les performances et minimiser les coûts opérationnels. Nous considérons les systèmes de file d'attente comme un modèle pour les infrastructures cloud IaaS et apportons des méthodes d'apprentissage pour allouer efficacement le bon nombre de ressources. Notre objectif est de minimiser une fonction de coût en tenant compte des coûts de performance et opérationnels. Nous utilisons différents types d'algorithmes de RL (du ≪ sans-modèle ≫ au modèle relationnel) pour apprendre la meilleure politique. L'apprentissage par renforcement s'intéresse à la manière dont un agent doit agir dans un environnement pour maximiser une récompense cumulative. Nous développons d'abord un modèle de files d'attente d'un système cloud avec un nœud physique hébergeant plusieurs ressources virtuelles. Dans cette première partie, nous supposons que l'agent connaît le modèle (dynamiques de l'environnement et coût), ce qui lui donne la possibilité d'utiliser des méthodes de programmation dynamique pour le calcul de la politique optimale. Puisque le modèle est connu dans cette partie, nous nous concentrons également sur les propriétés des politiques optimales, qui sont des règles basées sur les seuils et l'hystérésis. Cela nous permet d'intégrer la propriété structurelle des politiques dans les algorithmes MDP. Après avoir fourni un modèle de cloud concret avec des arrivées exponentielles avec des intensités réelles et des données d'énergie pour le fournisseur de cloud, nous comparons dans cette première approche l'efficacité et le temps de calcul des algorithmes MDP par rapport aux heuristiques construites sur les distributions stationnaires de la chaîne de Markov des files d'attente. Dans une deuxième partie, nous considérons que l'agent n'a pas accès au modèle de l'environnement et nous concentrons notre travail sur les techniques de RL. Nous évaluons d'abord des méthodes basées sur un modèle où l'agent peut réutiliser son expérience pour mettre à jour sa fonction de valeur. Nous considérons également des techniques de MDP en ligne où l'agent autonome approxime le modèle pour effectuer une programmation dynamique. Cette partie est évaluée dans un environnement plus large avec deux nœuds physiques en tandem et nous évaluons le temps de convergence et la précision des différentes méthodes, principalement les techniques basées sur un modèle par rapport aux méthodes sans modèle de l'état de l'art. La dernière partie se concentre sur les techniques de RL basées sur des modèles avec une structure relationnelle entre les variables d'état. Comme ces réseaux en tandem ont des propriétés structurelles dues à la forme de l'infrastructure, nous intégrons les approches factorisées et causales aux méthodes de RL pour inclure cette connaissance. Nous fournissons à l'agent une connaissance relationnelle de l'environnement qui lui permet de comprendre comment les variables sont reliées. L'objectif principal est d'accélérer la convergence : d'abord avec une représentation plus compacte avec la factorisation où nous concevons un algorithme en ligne de MDP factorisé que nous comparons avec des algorithmes de RL sans modèle et basés sur un modèle ; ensuite en intégrant le raisonnement causal et contrefactuel qui peut traiter les environnements avec des observations partielles et des facteurs de confusion non observés.

Title : Model-Based Reinforcement Learning for Dynamic Resource Allocation in Cloud Environments

Keywords : Reinforcement Learning, Auto-scaling policies, Cloud, Hysteresis, Factored reinforcement learning, Causal reinforcement learning

Abstract : The emergence of new technologies (Internet of Things, smart cities, autonomous vehicles, health, industrial automation, ...) requires efficient resource allocation to satisfy the demand. These new offers are compatible with new 5G network infrastructure since it can provide low latency and reliability. However, these new needs require high computational power to fulfill the demand, implying more energy consumption in particular in cloud infrastructures and more particularly in data centers. Therefore, it is critical to find new solutions that can satisfy these needs still reducing the power usage of resources in cloud environments. In this thesis we propose and compare new AI solutions (Reinforcement Learning) to orchestrate virtual resources in virtual network environments such that performances are guaranteed and operational costs are minimised. We consider queuing systems as a model for clouds IaaS infrastructures and bring learning methodologies to efficiently allocate the right number of resources for the users. Our objective is to minimise a cost function considering performance costs and operational costs. We go through different types of reinforcement learning algorithms (from model-free to relational model-based) to learn the best policy. Reinforcement learning is concerned with how a software agent ought to take actions in an environment to maximise some cumulative reward. We first develop queuing model of a cloud system with one physical node hosting several virtual resources. On this first part we assume the agent perfectly knows the model (dynamics of the environment and the cost function), giving him the opportunity to perform dynamic programming methods for optimal policy computation. Since the model is known in this part, we also concentrate on the properties of the optimal policies, which are threshold-based and hysteresis-based rules. This allows us to integrate the structural property of the policies into MDP algorithms. After providing a concrete cloud model with exponential arrivals with real intensities and energy data for cloud provider, we compare in this first approach efficiency and time computation of MDP algorithms against heuristics built on top of the queuing Markov Chain stationary distributions. In a second part we consider that the agent does not have access to the model of the environment and concentrate our work with reinforcement learning techniques, especially model-based reinforcement learning. We first develop model-based reinforcement learning methods where the agent can re-use its experience replay to update its value function. We also consider MDP online techniques where the autonomous agent approximates environment model to perform dynamic programming. This part is evaluated in a larger network environment with two physical nodes in tandem and we assess convergence time and accuracy of different reinforcement learning methods, mainly model-based techniques versus the state-of-the-art model-free methods (e.g. Q-Learning). The last part focuses on model-based reinforcement learning techniques with relational structure between environment variables. As these tandem networks have structural properties due to their infrastructure shape, we investigate factored and causal approaches built-in reinforcement learning methods to integrate this information. We provide the autonomous agent with a relational knowledge of the environment where it can understand how variables are related to each other. The main goal is to accelerate convergence by : first having a more compact representation with factorisation where we devise a factored MDP online algorithm that we evaluate and compare with model-free and model-based reinforcement learning algorithms ; second integrating causal and counterfactual reasoning that can tackle environments with partial observations and unobserved confounders.

Institut Polytechnique de Paris
91120 Palaiseau, France