Deep multi-agent reinforcement learning for dynamic and stochastic vehicle routing problems

Guillaume Bono

N° d'ordre NNT : 2020LYSEI096

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
l'INSA de LYON

Ecole Doctorale 512
InfoMaths

Spécialité/discipline de doctorat : Informatique

Soutenue publiquement le 28/10/2020, par :
Guillaume Bono

Deep Multi-Agent Reinforcement Learning for Dynamic and Stochastic Vehicle Routing Problems

Devant le jury composé de :

Mandiau, René	Professeur	Université Polytechnique des Hauts de France	Président
Charpillet, François	Directeur de Recherche	INRIA Nancy Grand Est	Rapporteur
Billot, Romain	Professeur	IMT Atlantique	Rapporteur
Beynier, Aurélie	Maître de Conférence HDR	Sorbonne Université	Examinatrice
Wolf, Christian	Maître de Conférence HDR	INSA de Lyon	Examinateur
Simonin, Olivier	Professeur	INSA de Lyon	Directeur de thèse
Dibangoye, Jilles	Maître de Conférence	INSA de Lyon	Co-encadrant
Matignon, Laëtitia	Maître de Conférence	Université Lyon 1	Co-encadrante
Pereyron, Florian	Ingénieur	Volvo Group	Invité

Département FEDORA – INSA Lyon - Ecoles Doctorales – Quinquennal 2016-2020

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** http://www.edchimie-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr INSA : R. GOURDON	**M. Stéphane DANIELE** Institut de recherches sur la catalyse et l'environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 Avenue Albert EINSTEIN 69 626 Villeurbanne CEDEX directeur@edchimie-lyon.fr
E.E.A.	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** http://edeea.ec-lyon.fr Sec. : M.C. HAVGOUDOUKIAN ecole-doctorale.eea@ec-lyon.fr	**M. Gérard SCORLETTI** École Centrale de Lyon 36 Avenue Guy DE COLLONGUE 69 134 Écully Tél : 04.72.18.60.97 Fax 04.78.43.37.17 gerard.scorletti@ec-lyon.fr
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** http://e2m2.universite-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : H. CHARLES secretariat.e2m2@univ-lyon1.fr	**M. Philippe NORMAND** UMR 5557 Lab. d'Ecologie Microbienne Université Claude Bernard Lyon 1 Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69 622 Villeurbanne CEDEX philippe.normand@univ-lyon1.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** http://www.ediss-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : M. LAGARDE secretariat.ediss@univ-lyon1.fr	**Mme Sylvie RICARD-BLUM** Institut de Chimie et Biochimie Moléculaires et Supramoléculaires (ICBMS) - UMR 5246 CNRS - Université Lyon 1 Bâtiment Curien - 3ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tel : +33(0)4 72 44 82 32 sylvie.ricard-blum@univ-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** http://edinfomaths.universite-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	**M. Hamamache KHEDDOUCI** Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tel : 04.72.44.83.69 hamamache.kheddouci@univ-lyon1.fr
Matériaux	**MATÉRIAUX DE LYON** http://ed34.universite-lyon.fr Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction ed.materiaux@insa-lyon.fr	**M. Jean-Yves BUFFIÈRE** INSA de Lyon MATEIS - Bât. Saint-Exupéry 7 Avenue Jean CAPELLE 69 621 Villeurbanne CEDEX Tél : 04.72.43.71.70 Fax : 04.72.43.85.28 jean-yves.buffiere@insa-lyon.fr
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** http://edmega.universite-lyon.fr Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction mega@insa-lyon.fr	**M. Jocelyn BONJOUR** INSA de Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69 621 Villeurbanne CEDEX jocelyn.bonjour@insa-lyon.fr
ScSo	**ScSo*** http://ed483.univ-lyon2.fr Sec. : Véronique GUICHARD INSA : J.Y. TOUSSAINT Tél : 04.78.69.72.76 veronique.cervantes@univ-lyon2.fr	**M. Christian MONTES** Université Lyon 2 86 Rue Pasteur 69 365 Lyon CEDEX 07 christian.montes@univ-lyon2.fr

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Résumé

Dans des environnements urbains en pleine extension, dans lesquels les conditions de circulation peuvent rapidement évoluer et augmenter les temps de trajet de façon imprévisible, il est très difficile de planifier les tournées des véhicules de livraisons assurant la logistique des entreprises, des commerces, et des particuliers. Les besoins logistiques de ces différents acteurs ont d'ailleurs tendance à se complexifier, avec l'apparition des services de livraison de nourriture, de livraison dans la même journée, ou de partage de véhicules. Pour faire face à de tels environnements dynamiques et stochastiques, il faut déployer des stratégies robustes et flexibles. Ces problèmes logistiques sont souvent modélisés en utilisant le formalisme des problèmes de planification de tournées de véhicules dynamique et stochastique (abrégés DS-VRPs en anglais). Ils peuvent être augmentés d'une large sélection de contraintes opérationnelles comme la capacité limitée des véhicules, ou les fenêtres temporelles dans lesquelles les clients préfèrent être servis (DS-CVRPTWs). La littérature de la recherche opérationnelle (OR) est riche de nombreuses approches heuristiques pour résoudre de tels problèmes dynamiques et stochastiques. Néanmoins celles-ci consistent la plupart du temps à considérer de nouveaux problèmes aux caractéristiques figées à chaque fois qu'un événement fait évoluer l'information disponible, en initialisant la recherche de solutions à partir de celles trouvées précédemment. Ceci permet de tirer avantages de toutes les heuristiques et méta-heuristiques existantes pour le problème de planification de tournées dans sa version la plus simple, statique et déterministe, qui n'ont eu de cesse de s'améliorer depuis les années 60, et permettent aujourd'hui d'obtenir des tournées très efficaces. Cependant, cela ne permet pas de capturer la dynamique des DS-VRPs et d'anticiper sur tous les futurs événements possibles.

Dans la littérature de l'apprentissage par renforcement (RL), il existe plusieurs modèles basés sur les processus de décision de Markov (MDPs) qui conviennent naturellement à décrire l'évolution de systèmes dynamiques et stochastiques. Leur application à des cas de plus en plus complexes est en plein essor grâce aux progrès réalisés dans le domaine des réseaux de neurones profonds (DNNs). Ces derniers sont aujourd'hui capables d'apprendre à fournir des approximations d'un grand nombre de classes de fonctions, dans des espaces de grandes dimensions, permettant ainsi de représenter les solutions de problèmes difficiles avec de très bonnes performances. Le défi scientifique principal dans l'application du RL à des problèmes combinatoires tels que le VRP est de trouver une structure d'approximation suffisamment riche, tout en restant compacte et efficace pour représenter l'état du système à tout instant. En ce qui nous concerne, les travaux les plus récents ayant étudié la possibilité d'appliquer des approches à base de DNNs au VRP ont montré que les architectures basées sur des mécanismes d'attention (AM) avait la capacité de représenter efficacement des ensembles de clients et leurs caractéristiques pour produire des règles de décision qui se généralisent à différents scénarios et différentes configurations du problème. Néanmoins, à notre connaissance, aucune des approches existantes utilisant des DNNs ne tient compte du caractère multi-agent du problème. Ils utilisent en effet une reformulation dans lequel un seul véhicule réalise l'ensemble des livraisons en l'autorisant à revenir au

dépôt central autant de fois que nécessaire pour ne pas enfreindre sa limite de capacité. Étant développée dans ce cadre, les architectures existantes n'ont donc pas la capacité de représenter l'état d'une flotte de véhicules déployés en parallèle, et ne peuvent donc pas leur fournir des règles de décisions adaptées au cours de leurs tournées.

Dans cette thèse, nous avons étudié comment appliquer les méthodes issues de l'apprentissage par renforcement profond (Deep RL) aux DS-VRPs soumis à des contraintes opérationnelles variées, en tenant compte du caractère multi-agent de ces problèmes. Dans un premier temps, nous avons exploré les méthodes basées sur le gradient de la politique dans le cadre du RL multi-agent (MARL). Nous avons contribué à étendre les fondements mathématiques des approches acteur-critique (AC) dans le cas des MDPs partiellement observables décentralisés (Dec-POMDPs). Nous nous sommes basés sur un paradigme d'entraînement assez original en apprentissage (où les modèles de transition et de récompense sont inconnus) mais ayant fait ses preuves en planification (où les modèles sont connus). Dans ce paradigme que nous appelons "entraînement centralisé pour un contrôle décentralisé" (CTDC), on donne accès à l'ensemble de l'information disponible en regroupant les observations de tous les agents, ainsi qu'à l'état caché du système pendant l'entraînement. On veille néanmoins à préserver la capacité des agents à choisir leurs actions de façon décentralisée, en se basant uniquement sur leurs propres observations individuelles. Nous avons évalué notre approche par rapport à deux autres paradigmes proposant un apprentissage complètement centralisé ou complètement décentralisé. Nos expérimentations montrent l'intérêt d'une telle approche, qui a plus de facilités à explorer l'espace de solution que la version complètement centralisé, tout en bénéficiant de l'information jointe disponible pour améliorer l'efficacité de l'entraînetment par rapport à la version complètement décentralisée.

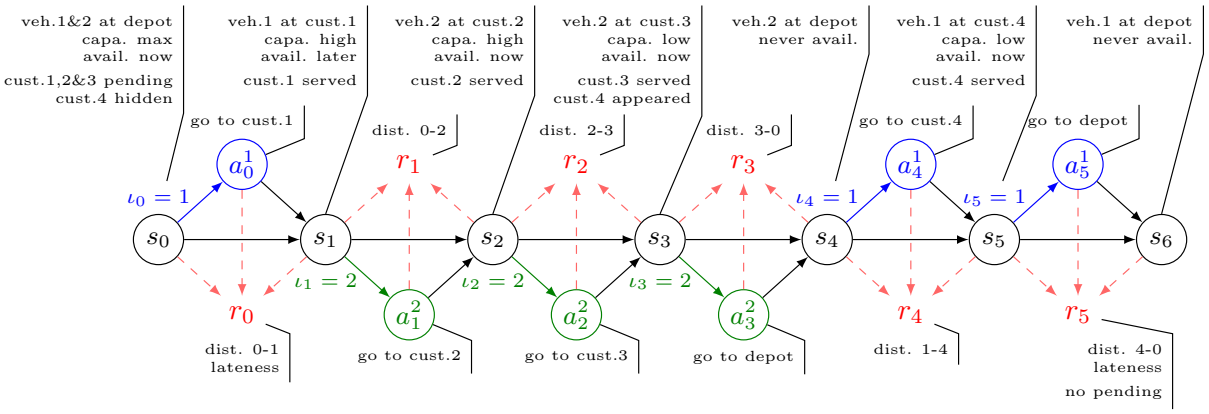


FIGURE 1 – Exemple de trajectoire échantillonnée avec notre modèle sMMDP d'un DS-VRP jouet à quatre clients et deux véhicules

Pour résoudre des problèmes de planification de tournées dynamique et stochastique, nous avons ensuite développé un nouveau modèle de MDP multi-agent séquentiel (sMMDP) dans lequel les agents ont une observation parfaite de l'état de l'environnement. Ce qui fait l'intérêt et l'originalité de ce nouveau modèle est la séquentialité des actions individuelles. Cette caractéristique implique que les effets des actions de chaque agent sur l'état du système et la récompense globale perçue peuvent être isolés. Nous avons formalisé ce modèle et établi les propriétés permettant de construire les algorithmes de résolution similaire aux autres variantes de MDPs. Nous avons ensuite comparé ce modèle à son équivalent en MDP multi-agent (MMDP) où la prise de décision est simultanée, et évalué leurs performances relatives sur un problème de

planification de tournée jouet. Pour de tels problèmes, qui vérifient la propriété de séquentialité des actions individuelles des agents, notre modèle sMMDP a un net avantage en termes d'efficacité d'entraînement par rapport au modèle simultané équivalent.

Ce cadre nous a alors permis de modéliser des DS-VRPs riches sous la forme de sMMDPs. Un exemple de trajectoire échantillonnée via ce modèle est donnée en Figure 1. Chaque véhicule est un agent indépendent ayant son propre état individuel. Tous les véhicules partagent l'information sur les clients en attente de service via un état commun. Leurs actions individuelles consistent à aller servir l'un des clients, ou à retourner au dépot pour mettre fin à leur tournée. Dans notre modèle, on considère que l'on peut prédire quand le véhicule aura terminé le service lié à sa dernière action et ainsi mettre à jour son état individuel instantanément sans impacter l'état des autres agents. Le signal de récompense perçu correspond à la distance parcourue par le véhicule, à laquelle s'ajoute une pénalité de retard s'il arrive à destination après la fenêtre de service du client. Une récompense globale est accordée à l'ensemble de la flotte à la fin de la trajectoire pour pénaliser les agents s'il reste des clients n'ayant pas été servis.

Afin de permettre le passage à l'échelle et de pouvoir dépasser le stade de problèmes jouets, nous avons proposé une nouvelle architecture de réseau de neurones à base de mécanismes d'attention que nous avons baptisé MARDAM. Cette architecture s'organise autour de quatre blocs afin de représenter les différents facteurs composant l'état du système et correspondant aux clients et aux véhicules. Elle fournit une règle de décision à chaque agent à son tour étant donné l'état global du système.

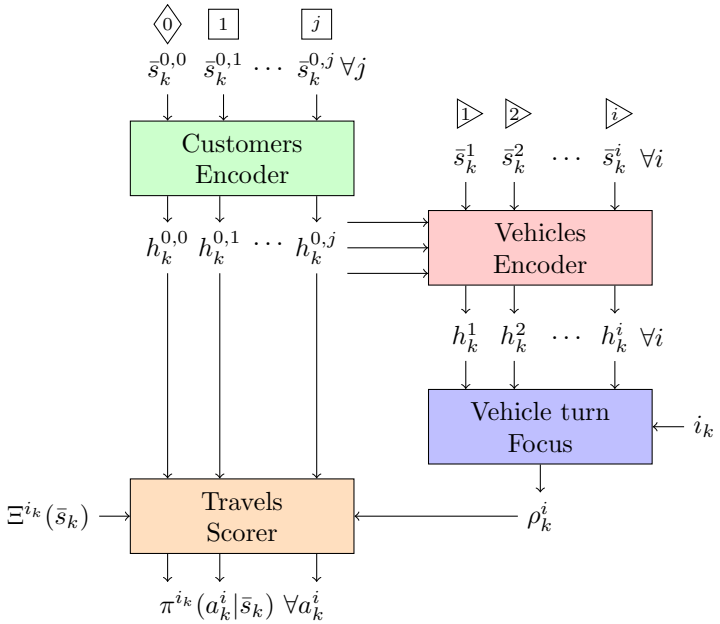


FIGURE 2 – Architecture globale de notre réseau de neurones MARDAM servant de politique paramétrée pour les DS-VRPs modélisés sous la forme de sMMDPs.

Les quatre blocs composant notre politique représentés sur la Figure 2 se répartissent la fonction de représentation de l'état du système comme suit. Le flot d'information au sein du réseau va de haut en bas. Le bloc "Customer Encoder" est le premier point d'entrée de notre réseau. Il a pour rôle de représenter l'ensemble des clients avec leurs caractéristiques. Il doit apprendre à extraire les patterns qu'ils forment les uns par rapport aux autres pour permettre à MARDAM de généraliser à différentes configurations du problème. Il fournit le contexte dans lequel les véhicules évoluent. Le second bloc "Vehicle Encoder" s'appuie sur les représentations

des clients produites par le premier pour extraire une représentation interne de chaque véhicule dans la flotte à partir de son état individuel. Le troisième bloc “Vehicle turn focus” focalise ces représentations de tous les véhicules de la flotte sur celui dont c'est le tour de choisir une nouvelle action, après qu'il ait fini de servir le client choisi précédemment. Nous obtenons ainsi une représentation de l'état interne du véhicule ayant à prendre une décision. Enfin, cet état interne est mis en relation avec la représentation de chaque client pour obtenir un score pour chaque action dans le bloc “Travels scorer”. Un masque est appliqué à ces scores pour empêcher le véhicule de choisir un client déjà servi, ou dont la demande excède sa capacité restante. On obtient ainsi, après normalisation, une règle de décision pour l'agent en cours donnant la probabilité de choisir chaque client comme prochaine destination.

Enfin, nous avons développé un ensemble de bancs de tests artificiels pour évaluer les performances de MARDAM sur plusieurs aspects des problèmes de planifications de tournées dynamiques et stochastiques. Nous avons empiriquement démontré sa capacité à généraliser à différentes configurations de clients, à maintenir des tournées robustes face à des temps de trajets incertains, et à s'adapter dynamiquement à des clients apparaissant en cours de tournées. De plus, la qualité des tournées obtenues est compétitive par rapport aux approches existantes. Afin d'évaluer MARDAM dans des situations plus réalistes, nous avons aussi dévelopé un autre banc de test basé sur une simulation microscopique de traffic permettant de simuler un large ensemble de véhicule sur un réseau routier. Nous présentons les résultats préliminaire issus de l'entraînement de MARDAM dans cet environnement, et mettons en avant sa capacité à s'adapter à différents réseaux, différents scénarii de traffic, et différentes configurations de clients.

Abstract

Routing delivery vehicles in dynamic and uncertain environments like dense city centers is a challenging task, which requires robustness and flexibility. Such logistic problems are usually formalized as Dynamic and Stochastic Vehicle Routing Problems (DS-VRPs) with a variety of additional operational constraints, such as Capacitated vehicles or Time Windows (DS-CVRPTWs). Main heuristic approaches to dynamic and stochastic problems in the OR literature often simply consist in restarting the optimization process on a frozen (static and deterministic) version of the problem given the new information available based on the previous solutions found. It makes it possible to re-use all hand-crafted expert heuristic and meta-heuristic for VRPs, which have been shown to output high-quality routes but might fail to properly capture and anticipate the dynamics of DS-VRPs.

Reinforcement Learning (RL) offers models such as Markov Decision Processes (MDPs) which naturally capture the evolution of stochastic and dynamic systems. Their application to more and more complex problems has been facilitated by recent progresses in Deep Neural Networks (DNNs), which can learn to represent a large class of functions in high dimensional spaces to approximate solutions with very high performances. Finding an efficient, compact and sufficiently expressive state representation is the key challenge in applying RL to complex combinatorial problems such as VRPs. Recent work exploring this novel approach demonstrated the capabilities of Attention Mechanisms to represent sets of customers and learn policies generalizing to different scenarios and configurations. However, to our knowledge, all existing work using DNNs reframe the VRP as a single-vehicle problem and cannot provide online decision rules for a fleet of vehicles.

In this thesis, we study how to apply Deep RL methods to rich DS-VRPs as multi-agent systems. We started by exploring the class of policy-based approaches in Multi-Agent RL, and extended the mathematical foundation of Actor-Critic methods for Decentralized, Partially Observable MDPs (Dec-POMDPs) to a somewhat new paradigm inspired by planning called Centralized Training for Decentralized Control (CTDC). In this CTDC paradigm, we consider that we have access to the complete joint information during training, while preserving the ability of agents to choose their individual actions in a decentralized way. We presented results highlighting the advantages of the CTDC paradigm compared to fully centralized and fully decentralized approaches.

To address DS-VRPs, we then introduce a new sequential multi-agent model we call sMMDP. This fully observable model is designed to capture the fact that consequences of decisions can be predicted in isolation. Once we established the classical Bellman equations, we experimented on its performance compared to the existing MMDP. Afterwards, we used it to model a rich

DS-VRP. To represent the state of the customers and the vehicles in this new model, we designed a new modular policy network we called MARDAM. It provides online decision rules adapted to the information contained in the state, and takes advantage of the structural properties of the model.

Finally, we developed a set of artificial benchmarks to evaluate the flexibility, the robustness and the generalization capabilities of MARDAM. We maintain competitive results compared to more traditional OR heuristics, and report promising results in the dynamic and stochastic case. We demonstrated the capacity of MARDAM to address varying scenarios with no re-optimization, adapting to new customers and unexpected delays caused by stochastic travel times. We also implemented an additional benchmark based on micro-traffic simulation to better capture the dynamics of a real city and its road infrastructures. We reported preliminary results as a proof of concept that MARDAM can learn to represent different scenarios, handle varying traffic conditions, and customers configurations.

Acknowledgements

First, I would like to thank my thesis director, Olivier Simonin, for the coordinating and counseling role he played throughout these four years. He helped conciliating in periods of tension during this thesis, and was a key advisor to lead it towards what it became today.

I also address my sincere thanks to my thesis supervisor, Jilles Dibangoye, who I respect for his sheer scientific commitment. Our professional relation went through ups and downs but he had the patience and wiseness to get over these tense periods, enabling a constructive collaboration since then.

I thank my two other supervisors, Laëtitia Matignon and Florian Pereyron, who provided me with helpful advices and feedbacks on the work I conducted.

I want to thank the Volvo / INSA Lyon chair for initiating and financing my work for the first three years of this thesis.

I am grateful to all the members of the jury, François Charpillet, Romain Billot, Aurélie Beynier, René Mandiau, and Christian Wolf, for the interest they gave to my work, and the time they offered me. I would like to address special thanks to François Charpillet who had an eye on this thesis since its first year as a member of the following committee.

I would like to acknowledge all coffee and crossword addicts, board games players, and other members of the CITI lab, which I had pleasure discussing of both scientific and non-scientific topics at lunch time throughout these years. I want to make a special mention to the fellow PhD students I shared my office with, Mihai, Jonathan, Patrick, Tristan and Gautier, for all the discussions and laughs we shared, and without whom this thesis would not have been as endurable as it was.

I am very grateful to my parents, who offered me a roof, and everything I ever needed to study in good conditions. Finally, to Marion, who gave me the love, strength and motivation I needed to successfully go through all these years.

List of Abbreviations

TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
PDP	Pickup and Delivery Problem
DARP	Dial-A-Ride Problem
CVRPTW	Capacitated VRP with Time Windows
DS-VRP	Dynamic and Stochastic VRP
DS-CVRPTW	Dynamic and Stochastic Capacitated VRP with Time Windows
MILP	Mixed Integer Linear Programming
CP	Constraint Programming
ADP	Approximate Dynamic Programming
VNS	Variable Neighborhood Search
LNS	Large Neighborhood Search
ACO	Ant Colony Optimization
GA	Genetic Algorithm
RL	Reinforcement Learning
MARL	Multi-Agent RL
MDP	Markov Decision Process
MMDP	Multi-agent MDP
POMDP	Partially Observable MDP
Dec-POMDP	Decentralized POMDP
sMMDP	sequential MMDP
CTCC	Centralized Training for Centralized Control
DTDC	Decentralized Training for Decentralized Control
CTDC	Centralized Training for Decentralized Control
SGD	Stochastic Gradient Descent
DNN	Deep Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
MHA	Multi-Head Attention
AM	Attention Model
MARDAM	Multi-Agent Routing Deep Attention Mechanisms

List of Notations

X	Random variable
$\mathcal{X}$	Domain of the variable X
$x \in \mathcal{X}$	Realization of the variable X
$\Pr\{X = x\}$ $\Pr\{x\}$	Probability that the variable X takes value $x \in \mathcal{X}$
$\Pr\{y \mid x\}$	Conditional probability that variable Y takes value y given that $X = x$
$\mathrm{E}[X]$	Expected value of the random variable X
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$\mathcal{B}(p)$	Bernoulli distribution with probability of success p
$\mathcal{U}(a, b)$	Discrete uniform distribution ranging from a to b
x	Scalar value
$\boldsymbol{x}$	Tensor, matrix or vector
$[\boldsymbol{x}]_i$	Element i of vector $\boldsymbol{x}$
δ_y^x	Kronecker's delta $\delta_y^x = 1$ if $x = y$, 0 otherwise
$\wp(\mathcal{X})$	Set of all possible subsets, a.k.a. power-set, of a set $\mathcal{X}$
$[\![a, b]\!]$	Set of integers $i \in \mathbb{Z}$ such that $a \leqslant i \leqslant b$

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In our modern world, urban areas attract more and more inhabitants and economic activities. In 2017, cities, town and suburbs in Europe sheltered more than 70% of its population and in 2014, metropolitan regions generated 47% of its Gross Domestic Product (GDP) [21]. Moreover, customers consumption habits are evolving, and new services are emerging with the development of e-commerce, same-day delivery, externalized food delivery, or car-ride sharing. Such densely-populated environments with important economic actors require well-organized logistic distribution networks on schedules tighter than they have ever been. Developing complex logistic networks while respecting the social, economic and ecological objectives that condition our future is a challenging task for decision makers and all actors of the sector.

New technologies in the domains of smart cities and autonomous vehicles open up many opportunities to improve the capabilities of such logistic systems. Indeed, they give us access to real-time information on uncertain factors that can affect logistic operations, such as traffic conditions and parking space availability. They also offer the possibility to take decisions in a decentralized way, such that agent can react and adapt to local perturbations while still fulfilling a common objective.

The *Vehicle Routing Problem* (VRP) is the generic abstract problem class that models multi-vehicle logistics. It has been extensively studied since the late 50s [28], and numerous extensions have been proposed to take into account a variety of operational constraints [111], such as Capacitated Vehicles and Time Windows (CVRPTW). If classical approaches mainly focused on a static and deterministic formulation where customers and travel costs are known a-priori, variants of the problem integrating some *Dynamic and Stochastic* information (DS-VRPs) received increasing attention in the last decades [85, 90]. Solving such DS-VRPs is the main motivating problem for the work developed in this thesis.

Context of the thesis. This thesis was initiated and founded by the industrial chair held by the Volvo group and operated by INSA Lyon under the supervision of Didier Remond. Although the main activity and domain of expertise of Volvo are not logistic operations, they design and produce trucks covering the needs of logisticians ranging from long-haul freight to last-kilometer delivery, with more focus on the former. The thesis being prospective, potential impacts for Volvo will only appear on the middle- or long-term. Our contributions will support future development at Volvo as an early step towards services related to city logistics. Additionally, part of their research effort is directed towards improving the energetic efficiency of their vehicles. To that end, the team of Florian Pereyron, who participated in the supervision of this thesis, develops

realistic simulations and generative models which cover both the mechanical and energetic aspect of Volvo trucks, but also the traffic conditions encounter on city roads. We could use these simulation tools to sample delivery trajectories while evaluating and training our decision policy.

This thesis was carried out in the INRIA/INSA team *CHROMA*, more precisely in the work group hosted at the CITI laboratory of INSA Lyon. This group specializes in artificial intelligence for robotics and multi-agent systems, with contribution ranging from mathematical foundations of Multi-agent RL, to real-world applications on mobile robotics, including drones and autonomous vehicles.

Positionning of the thesis. For the past sixty years, the Operations Research community has been developing and refining its approaches to solve the problem of finding routes serving all customers for a fleet of vehicles minimizing the costs while respecting some constraints. Exact approaches such as Branch-and-Cut-and-Price [79] have reach a level of performances where they can find routes for problems with up to a few hundreds of customers and prove the optimality of the solutions. For larger problems, or ones with more complex sets of constraints, meta-heuristics search such as Simulated Annealing [82], Ant Colony Optimization [101] or Genetic Algorithms [68] have been successfully used to obtain near optimal results. They are combined with specialized local operators, like 2-opt [27] and its variations. However, the existing approaches for DS-VRPs mostly rely on the same hand-crafted expert heuristics that are used for static and deterministic problems, which do not take full advantage of all the information available, especially the past experiences from which the system could automatically learn specialized heuristics to generate near-optimal routes much faster online.

When decisions should be adapted dynamically to a stochastic environment, *Reinforcement Learning* (RL) [105] offers promising tools to optimize policies that prescribe actions conditioned on the state of the system in a sequential decision-making process. Exact RL methods scale poorly with the dimensions of the problem. That is why we need customized approximate representations designed such that the subset of solutions we can explore matches the structure of the problems. This subset is then more likely to contain near optimal solutions. Hopefully, the generalization power of Deep Neural Networks (DNNs) and their capability to learn complex non-linear functions have open many opportunities in the field of Deep RL [70, 71, 97], and more specifically their application to combinatorial problems [7]. Recent advances in Deep Learning over graphs provide state representations to encode VRPs [7, 47, 50]. They have given birth to fast and competitive heuristics that generalizes over a large set of problem instances. The main difficulties are to create meaningful vector representations of graph-structured data and to efficiently take into account the constraints of the problems. Nonetheless, all existing works in this domain reframe the VRP as a single-vehicle problem. This thesis contributes towards taking into account a multi-agent viewpoint. It also extends the class of problems that approaches based on Deep RL are capable of addressing to rich DS-VRPs. But to what extend this multi-agent viewpoint can help leveraging the issues encountered in DS-VRPs?

This is what we will try to answer in this thesis. The core of our approach is globally summarized by Figure 1.1. Two modules interact with one another in a standard RL loop. An environment, modelled as a sequential Multi-agent Markov Decision Process (sMMDP), is controlled by a policy, here implemented as a DNN architecture called Multi-Agent Routing Deep Attention Mechanisms (MARDAM). The policy observes the state of system, in this case describing the customers and the vehicles, from which it can choose an action. It is trained by a learning algorithm which updates the policy parameters based on a reward signal provided by the environment for each state transition provoked by an action.

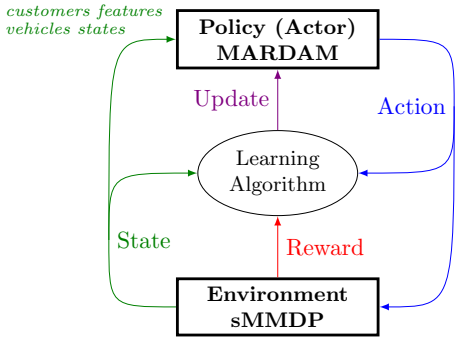


Figure 1.1 – General architecture of our approach. To model and address rich DS-VRPs, we propose a policy network, **MARDAM**, implemented using attention mechanisms, and an event-based, multi-agent MDP we call **sMMDP**. The different arrows illustrates the standard interaction and training loop of Reinforcement Learning.

The sMMDP model is a new variant of MMDP which we developed to take into account that agents do not act simultaneously. MARDAM is also a novel DNN architecture that can learn to provide decision rules to the vehicles depending on the current state of the system. This state captures both the variability of the customers configurations the fleet can face, and the uncertain and dynamic information the vehicles collect while serving them. As the objective of the learning algorithm is to maximize the expected cumulated reward, the policy is trained to adapt to the information contained in the state to anticipate on the dynamic and stochastic evolution of the system.

Scientific challenges. To address DS-VRPs using Multi-agent Deep Reinforcement Learning, we face the following scientific challenges:

1. The basic VRP already has a non-polynomial time complexity, i.e. the VRP is NP-hard, which limits our capacity to scale up to high numbers of customers and vehicles. We did not focus on this issue in this thesis, and our approach is limited in term of scalability.

2. Optimally solving the dynamic and stochastic VRP would correspond to solving static VRPs for every possible realizations and evolutions of its random variables. It would quickly become untractable, hence we need robust and flexible approximate strategies to adapt our solutions to the data available at any time.

3. To – at least partially – transfer the decision authority from a centralized dispatcher to vehicles having their own individual decision capabilities, we need to study the influence of individual actions on the global objective, and identify some structure to exploit in our model.

4. To avoid training a DNN for a specific configuration of the DS-VRP, but instead allow it to generalize over multiple instances with no fine-tuning and costly retraining, we need to represent all the characteristic of the problem, i.e. a graph of customers evolving through time with stochastic edge weights.

5. Finally, training DNNs is a data-hungry task and it requires a large amount of trajectory samples. Hence we need rich dataset and benchmarks that capture the diversity of configurations and dynamic evolution our learned heuristic will have to face.

Outline of the manuscript. The following of this manuscript is organized as follow:

- In Chapter 2, we build a taxonomy of VRP variants we are interested in, and formalize them using a classical Mixed Integer Linear Programming (MILP) model. We review a few exact approaches and heuristics, then draw an overview of which methods were previously applied to which variant of the problem. Finally we introduce the basics of Deep Learning and review the state of the art methods based on DNNs, especially how graphs are being encoded to represent the customer configurations and states in a VRP.

- In Chapter 3, we introduce some background knowledge on Reinforcement Learning and Markov Decision Processes (MDPs), especially their multi-agent (MMDPs) and Decentralized, Partially Observable (Dec-POMDPs) extensions. We review the basics of existing resolution methods, including Value-based and Policy-based algorithms. Finally, we present some preliminary work we conducted on a training paradigm we refer to as Centralized Training for Decentralized Control (CTDC). In this paradigm, more information can be used during training than is available when executing the learned policy. We contribute to the mathematical foundation of Actor Critic (AC) methods in this paradigm, and evaluate our approach through experiments comparing it to other existing paradigms on standard Dec-POMDP benchmarks.

- In Chapter 4, we present a new variant of MMDPs we call a sequential MMDP (sMMDP), that can capture the properties of VRPs problems. We especially focus on the fact that agents take decisions independently triggered by specific events. We start by building the model iteratively through the properties that differentiate it from an MMDP, then derive theorems corresponding to Bellman equations for this new sequential variant. We evaluate the performances of our model compared to an MMDP, and show how it scales better with the number of agents.

- In Chapter 5, we explain how we model a Dynamic and Stochastic Capacitated VRP with Time Windows (DS-CVRPTW) as a sMMDP. We then describe how we designed MARDAM as a modular policy with blocks dedicated to each factor of the state. We discuss the properties we needed to efficiently represent sets of customers and vehicles, and the main difference with existing architectures that enables us to capture the multi-agent dimension of VRPs, and use MARDAM as an online policy in a dynamic and stochastic environment.

- In Chapter 6, we evaluate MARDAM on a set of artificially generated benchmarks. We demonstrate the capability of our policy to address a variety of scenarios, with no specific retraining. MARDAM is able to flexibly adapt to dynamically appearing customers, and compromise with delays caused by stochastic travel times. We also present a more complex benchmark based on a finer simulation of the dynamics of a road network. We got some preliminary results training MARDAM, and discuss how it could be further improve.

- Finally, Chapter 7 concludes the manuscript summarizing our contributions and discussing perspectives about the future of our approach in term of learning algorithms, model analysis, architecture improvement, and experimental validations.

Chapter 2

Vehicle Routing Problems

In an ongoing trend for globalization, our modern world relies heavily on complex multi-modal logistic networks. Planes, boats, trains, long-haul trucks, smaller delivery vehicles and even bikes work together to ship goods from one side of the world to the other. Their synchronization and the optimization of their costs require a lot of careful planning, and robust recourse solutions in case of incidents. Among other costs to optimize, at one end of the logistic chain, assigning and ordering the deliveries to the final customers is one of the big challenges in Operations Research (OR), with high social, ecological and economic stakes.

A standard mathematical abstraction of this combined assignment and ordering problem is called the *Vehicle Routing Problem* (VRP). To the best of our knowledge, it has been first introduced in 1959 by Dantzig and Ramser [28], who considered Capacitated vehicles (CVRP). Since then, VRPs have been thoroughly studied in the OR literature [111], and extended to take into account a variety of operational constraints, such as customers with Time Windows (VRPTW) or services including Pickup and Delivery (PDP). On top of these constraints, what usually makes the difference between classical abstractions and real-world applications is the evolution (Static or Dynamic) and the quality (Deterministic or Stochastic) of the information available, as stated by Pillac et al. [81].

Original static and deterministic VRPs are usually formalized as (Mixed) Integer Linear Programs ((M)ILPs) [78] or Constraint Optimization Programs (COPs) [101]. These are solved offline, either by exact methods such as Branch-and-Bound, or heuristics such as Large Neighborhood Search (LNS), Ant Colony Optimization (ACO) or Genetic Algorithms (GAs). In this static and deterministic setup, solutions take the form of plans which can then be executed by the dispatcher and the vehicles. Stochastic VRPs (S-VRPs) can also be addressed offline knowing the distributions governing the random variables of interest, and considering objectives with expected values or robustness criteria. Dynamic VRPs (D-VRPs) require a way to adapt the routes of the vehicles online, in reaction to newly available pieces of information. There are a few different methods to achieve this, including re-optimizing the plan every time new information is available, or having a policy that provides decision rules for any possible state encountered.

Recently, a new class of heuristics based on Deep Reinforcement Learning (DRL) has emerged [7]. Using the representation power Deep Neural Networks (DNNs) offer, the goal is to learn to solve combinatorial problems from large sets of data. Key challenges include: efficiently representing unordered sets of customers, taking into account hard constraints of the problem; and keeping track of individual internal state for every vehicle. Once trained, these DRL approaches

have been shown to provide solutions competitive with the ones obtained by hand-crafted expert heuristics. Furthermore, in exchange for the initial offline training time investment, they have a clear advantage over other heuristics both in term of running time at execution. Moreover, they can be easily parallelized for solutions sampling.

The following of this chapter is organized as follow. Section 2.1 introduces classical formulation of a few VRP variants. Sections 2.2 and 2.3 review some classical exact and heuristic methods to address routing problems. Section 2.4 proposes a review of a few recent references with a focus on dynamic and stochastic problems. Section 2.5 reviews recent methods based on DRL. The three first sections correspond to a review paper [3] we presented at JFPDA in 2017.

2.1 VRPs formalization

In this section, we will build up a complete formalization of VRPs as a MILP, mostly inspired by the one of Cordeau et al. [24]. Figure 2.1 provides an overview of the VRP variations we will present, with a selection of constraints of interest. We also list a few common examples of variables that can make the problem dynamic and/or stochastic.

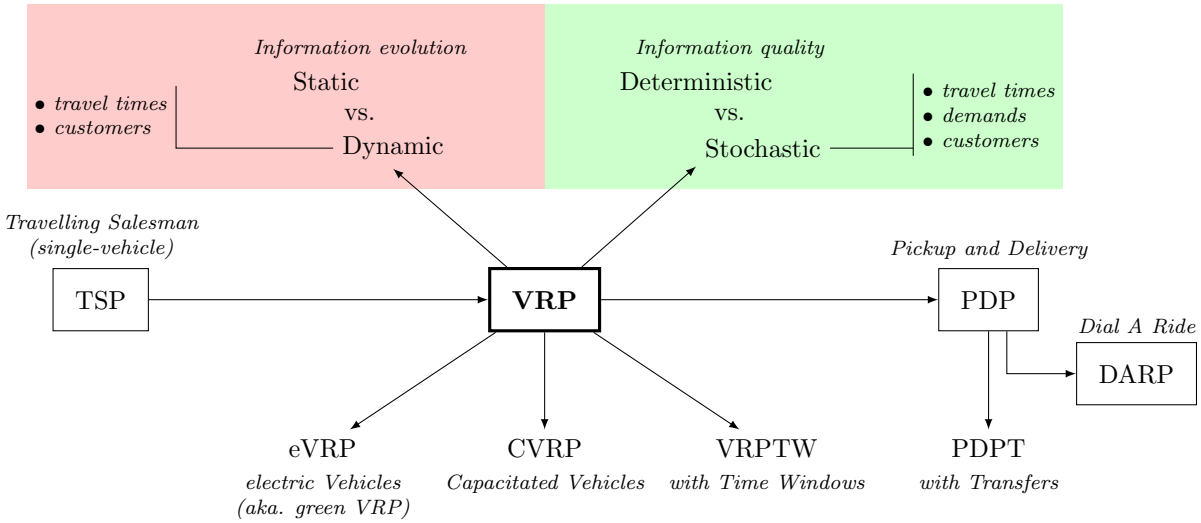


Figure 2.1 – Taxonomy of VRPs - In the middle row, framed nodes are variants of the problems. Below them are a selection of operational constraints that we consider addressing. Above them we illustrate a few common example of dynamic and/or stochastic elements.

We start with the historical Travelling Salesman Problem (TSP) which can be seen as a specialization of VRP for single-vehicle problems even though it was introduced way earlier in the literature. We then extend it to the simplest form of VRP, simply adding the multi-vehicle dimension. Afterwards we present a way to enforce the Capacity (CVRP), Time Windows (VRPTW) and energy (eVRP) constraints in a unified formalization using state variables. Finally we introduce the Pickup and Delivery Problem (PDP), and the closely related Dial-A-Ride Problem (DARP), for which we will consider Transfer (PDPT) possibilities.

2.1.1 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is a very well known problem that is closely related to VRPs. Its origin dates back to the 19$^{\text{th}}$ century, but is not clearly defined, as the problem has been discussed long before being mathematically formalized. Given a set of customers indexed by integers ranging from 1 to n which we note $[\![1, n]\!]$, and the costs c_{ij} of travelling from any customer i to any other customer j, the TSP consists in finding a tour of minimal total cost going through all customers once. It can be pictured as finding the Hamiltonian cycle of minimal cost in a fully-connected directed graph where each vertex i is a customer, and each edge i, j is weighted by the travel cost c_{ij}. Figure 2.2 illustrates an example of such graph, with the solution to the corresponding TSP.

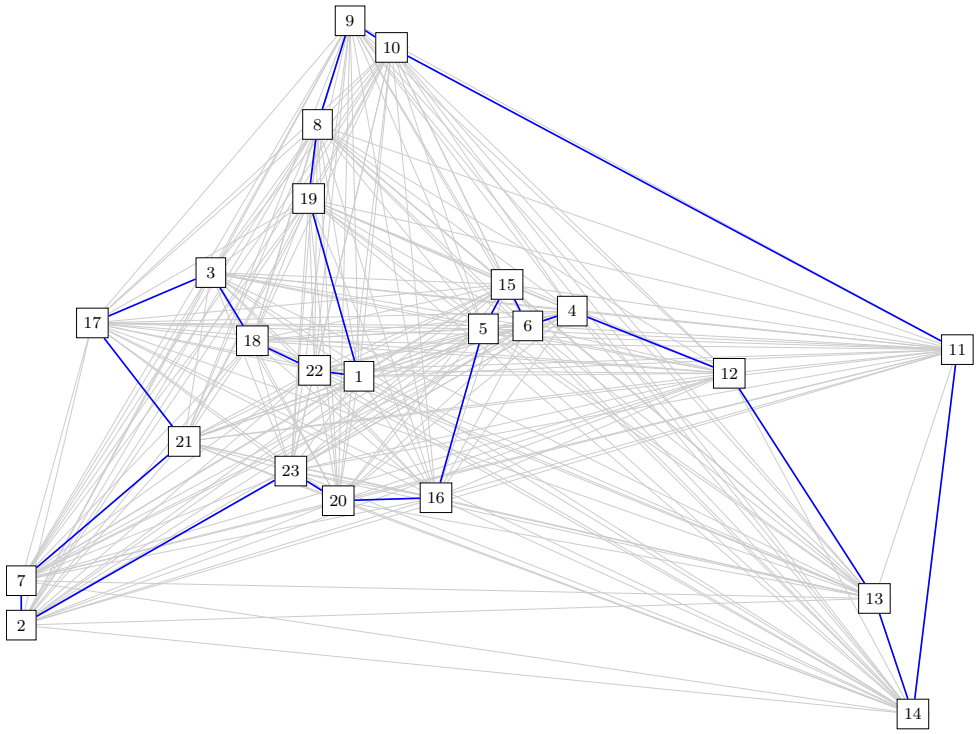


Figure 2.2 – Example of TSP instance with 23 customers, created based on the first nodes of the *berlin*52 instance in the TSPLIB [88] benchmark. Edges are weighted by the euclidean distance between the nodes, and the ones in blue form the optimal tour for this instance.

Mathematically, the problem can be formalized as ILP (2.1) with boolean decision variables $x_{ij} \in \{0,1\}$, indicating whether each edge i,j belongs to the route or not.

$$\min \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} c_{ij} \tag{2.1a}$$

$$\text{s.t.} \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1 \qquad \forall i \in [\![1,n]\!] \tag{2.1b}$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} - \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ji} = 0 \qquad \forall i \in [\![1,n]\!] \tag{2.1c}$$

$$\sum_{i\in S} \sum_{j\in S\setminus\{i\}} x_{ij} \leqslant |S| - 1 \qquad \forall S \subset [\![1,n]\!], |S| > 2 \tag{2.1d}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in [\![1,n]\!]\ \forall j \in [\![1,n]\!]\setminus\{i\} \tag{2.1e}$$

where (2.1a) encodes the objective of minimizing the total cost of the tour, (2.1b) constraints every node to be visited once and only once, (2.1c) enforces flux conservation, (2.1d) eliminates sub-tours, and (2.1e) defines the boolean decision variables.

The sub-tours elimination constraints (2.1d) are quite numerous ($\mathcal{O}(2^n)$). In practice, they are often iteratively added to the model only when a sub-tour is detected. The state consistency constraints we will consider in latter subsections will make these sub-tour elimination constraints redundant. Hence we will not develop further on their treatment.

Using a simple reduction of the Hamiltonian cycle problem, it is quite straightforward to prove that the TSP[1] is NP-complete. Even worse, without additional hypotheses, even the approximation problem, i.e. finding a solution within a proportional range above the optimal, is NP-hard. For example, a polynomial-time heuristic known as Christofides algorithm [19] provides a solution within 1.5 times the optimal value only for the euclidean TSP, where costs verify the triangular inequality $c_{ij} \leqslant c_{ik} + c_{kj}$.

2.1.2 Capacitated Vehicle Routing Problem

Next step to build the formalization of our rich VRP is to integrate its multi-agent dimension. We now consider a fleet of m vehicles to serve our n customers. We must assign customers to vehicles and route each vehicle through its assigned customers. We will consider a homogeneous fleet, meaning that all vehicles have identical characteristics, and suffer the same travel costs c_{ij}. They all initially start at a common depot, and must return to it at the end of their routes. This depot will be represented as two identical extra nodes 0 and $n+1$ in our graph of customers.

[1] Strictly speaking, the decision version of TSP, using a threshold on the objective value, is NP-complete.

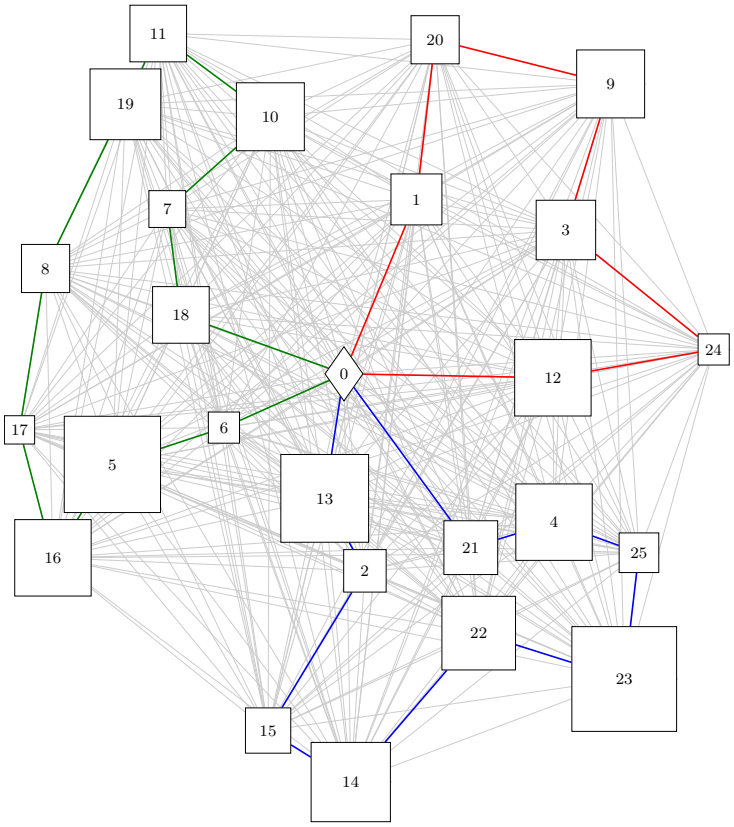


Figure 2.3 – Example of CVRP instance with 25 customers and a depot, created based on the first 26 nodes of the *test*2 instance in the VRPLIB [115] benchmark. Nodes are weighted by the demand of the customers, represented here by their size. As in previous example, edges are weighted by the euclidean distance between the nodes. The blue, green and red edges, show the 3 routes minimizing the total cost while respecting the capacity constraints.

To the best of our knowledge, the VRP was first introduced and formalized in [28]. To motivate the use of a fleet of vehicles, this original version implicitly includes capacity constraints. In this setup, more explicitly referred to as Capacitated VRP (CVRP), every customer j has a demand q^j corresponding to the quantity of items to deliver. Every vehicle has a limited capacity $\kappa_{\max}$ such that it can only serve a fraction of the customers. This abstract payload model does not account for the weights, volumes and shapes of the packages, but we will assume the demands are expressed in a standardized unit which factors these parameters in. Figure 2.3 illustrates an example of CVRP highlighting how customers are allocated to different vehicles to respect their capacity.

Formally, the boolean decision variables x_{ij}^k gain an extra index k to indicate which vehicle travel through edge i, j to serve customer j. We also introduce state variables κ_i^k to keep track of the remaining capacity of vehicle k before serving customer i (when it is assigned to it). Borrowing all other notations from the TSP, ILP (2.2) describes the VRP.

$$\min \sum_{k=1}^{m} \sum_{i=0}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n+1} x_{ij}^k c_{ij} \tag{2.2a}$$

$$\text{s.t.} \sum_{k=1}^{m} \sum_{\substack{j=1 \\ j\neq i}}^{n+1} x_{ij}^k = 1 \qquad \forall i \in [\![1, n]\!] \tag{2.2b}$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n+1} x_{ij}^k - \sum_{\substack{j=0 \\ j\neq i}}^{n} x_{ji}^k = 0 \qquad \forall k \in [\![1, m]\!] \ \forall i \in [\![1, n]\!] \tag{2.2c}$$

$$\sum_{j=1}^{n+1} x_{0j}^k \leqslant 1 \qquad \forall k \in [\![1, m]\!] \tag{2.2d}$$

$$x_{ij}^k(\kappa_i^k + q^i - \kappa_j^k) = 0 \qquad \forall k \in [\![1, m]\!] \ \forall i \in [\![0, n]\!] \ \forall j \in [\![1, n+1]\!] \backslash \{i\} \tag{2.2e}$$

$$\kappa_0^k = \kappa_{\max} \qquad \forall k \in [\![1, m]\!] \tag{2.2f}$$

$$\kappa_i^k \geqslant q^i \qquad \forall k \in [\![1, m]\!] \ \forall i \in [\![1, n]\!] \tag{2.2g}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall k \in [\![1, m]\!] \ \forall i \in [\![0, n]\!] \ \forall j \in [\![0, n]\!] \backslash \{i\} \tag{2.2h}$$

where (2.2a) states the objective of minimizing the cost of all routes, (2.2b) constraints every node to be visited once and only once by any vehicle, (2.2c) enforces flux conservation, (2.2d) guarantees that each vehicle can only leave the depot once, (2.2e) enforces capacity consistency from one customer to the next, (2.2f) sets the initial capacity at the depot, and (2.2g) prevents capacity overloads.

There exists an alternative 2-index formulation which is a direct adaptation of (2.1). It reduces the number of variables but increases the number of constraints, and cannot differentiate between individual vehicles. This way of handling capacity constraints is neither in the original formulation [28], nor in common CVRP ones [111], which generalize the sub-tour elimination constraints (2.1d) to guarantee that the total demand of any subset of customers S can be satisfied by incoming vehicles. In this form, we can see that, just as the TSP, the CVRP is also NP-hard.

Using state variables has the disadvantage of involving quadratic constraints (2.2e), which needs to be linearized or specifically handled as conditional constraint by solvers. It is however much closer to the MDP model we will present in Chapter 5, and falls into a unified formalization that can be used for many other constraints.

2.1.3 Unifying capacity, time windows, and energy constraints

State variables can be used to take into account a variety of operational constraints. We have already added variables κ_i^k to keep track of vehicles' capacities in CVRP. We now introduce new variables to address VRP with Time Windows (VRPTW) and electric VRP (eVRP, also known as green VRP).

In VRPTW, every customer j must be served for a duration d^j in a limited Time Window (TW) bounded by a ready time e^j (early) and a due time l^j (late). In addition to the travel costs c_{ij}, we are given the travel times τ_{ij} between pairs i, j of customers (plus the depot). As illustrated on Figure 2.4, these TW constraints can require some compromises on the travel cost to be met.

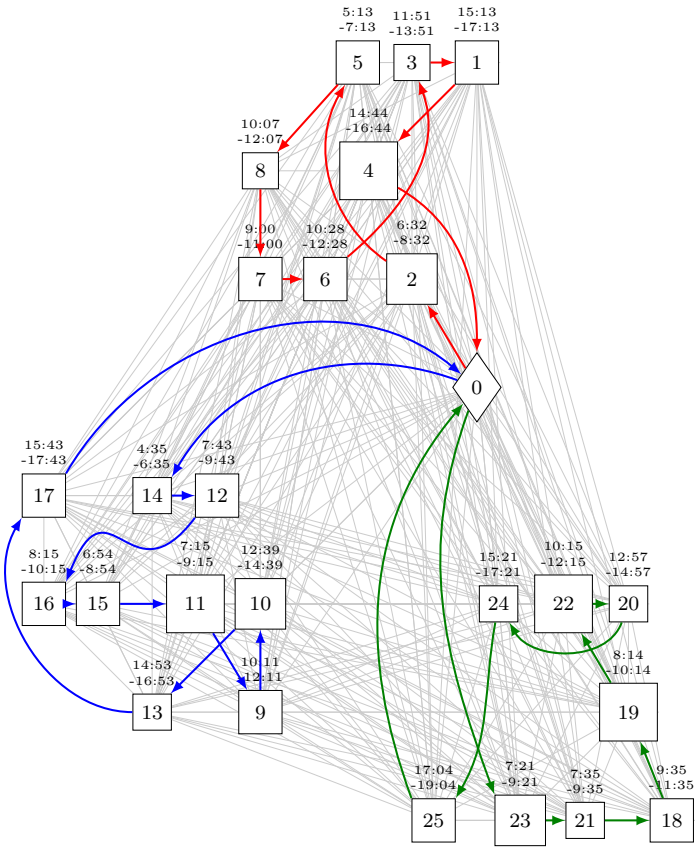


Figure 2.4 – Example of CVRPTW instance with 25 customers and a depot, created based on the first 26 nodes of the *rc*201 instance in the Solomon [102] benchmark. Nodes are weighted by the demand of the customers, represented here by their size, and labeled with the time window for this customer. The service duration is not represented. The euclidean distance between two nodes i and j, gives both the cost c_{ij} and the travel time τ_{ij} associated with the edge between them. The blue, green and red edges, show the 3 routes minimizing the total cost while respecting the capacity and time-window constraints.

We will not recall the entire ILP (2.2), but simply provide the 3 constraints to add to take into account TWs. We use state variables t_i^k to represent the time at which vehicle k starts serving customer i (if it is assigned to it):

$$x_{ij}^k(t_i^k + d^i + \tau_{ij} - t_j^k) \geqslant 0 \qquad \forall k \in [\![1, m]\!]\ \forall i \in [\![0, n]\!]\ \forall j \in [\![1, n+1]\!] \backslash \{i\} \tag{2.3a}$$

$$t_0^k = 0 \qquad \forall k \in [\![1, m]\!] \tag{2.3b}$$

$$e^j \leqslant t_j^k \leqslant l^j \qquad \forall k \in [\![1, m]\!]\ \forall j \in [\![1, n+1]\!] \tag{2.3c}$$

where (2.3a) maintains state consistency, (2.3b) initializes time when leaving the depot (arbitrarily at $t = 0$), and (2.3c) enforces TWs. Again, just as (2.2e), (2.3a) is quadratic and needs to be linearized or handled as a conditional constraint.

Very similarly, we can add many other dimensions to the problem. For example to take into account the energy consumption of electric vehicles, we can introduce state variables to keep track of the remaining charge of vehicles, an energy cost on every edge, and constraints to enforce state consistency, initialization and bounds. eVRP models [22, 36] often also imply additional charging station nodes which require more specific constraints.

2.1.4 Pickup and Delivery Problem

The last variant of routing problems we consider is the Pickup and Delivery Problem (PDP) [30], and its closely related application to passenger transport called Dial-A-Ride Problem (DARP) [84, 23]. In both these setups, every customer j is not longer associated with just a single node but a pair of nodes: a node labelled j for pickup associated with a $+q^j$ demand, and one for delivery labelled $n+1+j$ with a $-q^j$ demand. Obviously we need to constraint the routes such that the same vehicle handles the pickup and the delivery of a given customer, but also such that the pickup happens before the delivery.

2.1.5 Information evolution and quality

Many sources of uncertainty can be included in the model. Mostly three of them have been considered in the literature:

1. Stochastic Travel Times: $\{\tau_{ij}\ \forall i,j\}$ are random variables
2. Stochastic Demands: $\{q^j\ \forall j\}$ are random variables
3. Stochastic Customers: $\{\delta^j\ \forall j\}$ are new binary random variables that indicate whether customer j is present or not.

Whatever the source of uncertainty is, the problem is called Stochastic CVRPTW (S-CVRPTW). All approaches do not consider the same objective when dealing with a stochastic model. Some of them will consider a recourse action, for example returning to the depot if the customer's demand exceed the one that was expected. Some will try to minimize the risk of violating a time-window or a capacity constraint given the distributions of the random variables. Finally, other will add penalties to the objective for violated constraints and minimize its expected value.

Later, we will mainly focus on stochastic travel times and we choose to allow for relatively small time window violations by adding a proportional penalty term for any late service start at a customer. This is called *soft* Time Windows, by opposition to *hard* Time Windows when the constraints have no slack. Our objective becomes:

$$\min \sum_{k=1}^{m} \sum_{i=0}^{n} \sum_{\substack{j=0 \\ j\neq i}}^{n} x_{ij}^k \Big(c_{ij} + \kappa \, \mathrm{E}\left[(T_k^j - l^j)^+ \right] \Big) \tag{2.4}$$

where κ is the cost per unit time of lateness, $(x)^+ = \max\{0, x\}$, and the expected value E is taken over the distribution of random state variables T_j^k induced by the distribution of travel times τ_{ij} and constraints (2.3a)-(2.3b) that define transition rules. (2.3c) is modified to only constraint ready time:

$$t_k^i \geqslant e^i \qquad\qquad \forall k \in [\![1, m]\!]\ \forall i \in [\![0, n]\!] \tag{2.5}$$

2.1.6 Dynamic and Stochastic CVRPTW

The final variant of VRP we will introduce is Dynamic and Stochastic CVRPTW (DS-CVRPTW). In this problem, only a fraction of customers is known a-priori before the vehicles leave the depot. The remaining customers appear dynamically while the vehicles are travelling. The ratio between the number of unknown customers and the total number of customers is known as the *degree of dynamism* $r_{\text{dyn}} \in [0, 1]$ of the problem. A scenario based on a small toy problem is illustrated on Figure 2.5.

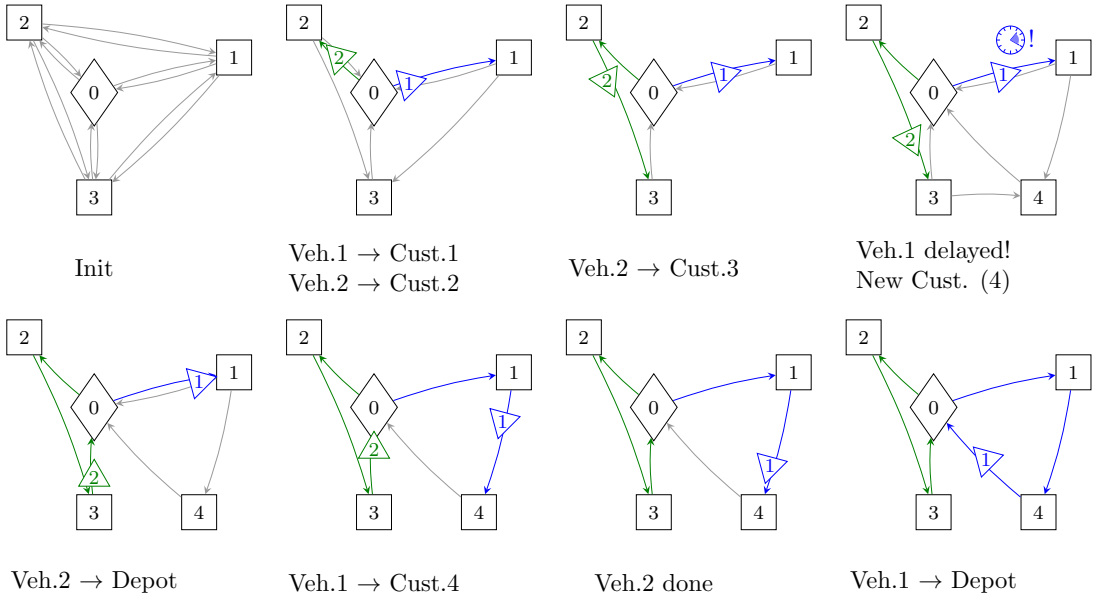


Figure 2.5 – Illustration of a small toy DS-CVRPTW. Two vehicles are stationed at depot 0. They are initially aware of three customers waiting to be served. While executing their initial plan, the environment evolves dynamically: delays can be experienced on some travels and new customers can appear. Routes are adapted online to get the best of the situation, compromising on delays, detours and customers satisfaction.

Most approaches relax the service constraint and allow to discard some customers requests. The ratio of the number of served customers on the total number of customers is called the *quality of service* of the solution. The way this quality of service is balanced with the cost of the solution depends on the approach. Not only do the different methods to address this problem have different objectives to optimize, they also differ in the form of the solution they provide. Some of them will anticipate and include some waiting points in the routes [94], relying on predefine strategies to serve newly appearing customers at proximity, which can be compared to some kind of complex recourse actions. Other methods will re-optimize the routes at every new customer encountered, sometimes maintaining a pool of virtual anticipatory plans to add some variety and speed up the search for a new solution [8]. Finally, closer to our approach and an MDP formulation, some methods will provide policies that take online measurements or observations of the state of the environment, and build the routes sequentially by committing only to one delivery at a time [98].

An Integer Linear Program is not anymore very well adapted to describe this variant of the problem. One could consider a new ILP every time a new customer appear, but would have to introduce additional state variables to take into account the deliveries that were already committed to and the partial routes traveled by the vehicles. We will later propose a model

inspired by Markov Decision Processes (MDP), which seems more adapted to DS-CVRPTW, as it naturally describe the temporal evolution of a stochastic environment.

2.2 Exact methods for solving VRPs

Now that we have presented and formalized the problem and a few of its variants, we will review some classical methods to solve it to optimality. A lot of engineering went into the development of powerful solvers, which combines many heuristics to quickly search the solution space. Our goal in this section is not to review in details all these optimizations, but to give an overview of the base methods involved.

2.2.1 Integer Linear Programming

These approaches directly address the problem using the ILP formulation introduced above. The goal is to find an assignment of the binary decision variable $x_{ij}^k \in \{0, 1\}$ minimizing the objective while satisfying constraints. Most algorithms rely on a Divide-and-Conquer approach and split the initial solution space into smaller sets. One of the basic algorithmic structure implementing this approach is called Branch-and-Bound [54, 57] and is described in Algorithm 2.1. The main idea behind this algorithm is that a fast heuristic `ComputeLowerBound` can quickly filter out subsets of the solution space that cannot lead to an improvement of the cost. The key point is the branching function `Branch` that subdivides the problem, and governs the branching factor and the depth of the search tree.

```
1  currentBest, upperBound ← FindInitialSolution(problem)
2  add problem to queue
3  while queue not empty do
4      problem ← pop item from queue
5      foreach subProblem in Branch(problem) do
6          if CanSolve(subProblem) then
7              candidate, cost ← Solve(subProblem)
8              if cost ⩽ upperBound then
9                  currentBest, upperBound ← candidate, cost
10         else if ComputeLowerBound(subProblem) ⩽ upperBound then
11             add subProblem to queue
```

Algorithm 2.1: Branch-and-Bound

One of the simplest way to implement the lower-bound heuristic is to linearly relax the ILP into a (convex) Linear Program (LP), which can be solved in $O(n^3)$ by the simplex algorithm for example. This relaxation consists in removing the integral constraint on the decision variable x_{ij}^k and allow them to take any real value in $[0, 1]$. The optimal cost of the relaxed LP is a lower-bound of the cost of the original ILP. However, the tighter the bound is, the less branches will be expanded, and this one can be quite loose. Advanced heuristics can be used as bounding function, but they need to be faster than expanding the branch itself to be efficient.

One example of branching function is to use the solution of the LP relaxation to identify the "most fractional" variable assignments ($x_{ij}^k \approx 0.5$) and create constraints that prevent it (i.e.

one branch will have $x_{ij}^k = 0$ as an additional constraint, the other $x_{ij}^k = 1$). More complex cutting planes [41], can be deduced from the LP solution to isolate the fractional solution from any integer solution. They help getting closer to integral solution before branching, leading to what is called the Branch-and-Cut algorithm [77]. Also, similarly to bounding functions, many specialized branching strategies based on expert-knowledge are available for specific problems. One simple example for VRPs is to branch on the number of vehicles involved in the solution.

Another optimization that can be conducted on this general class of Branch-and-Bound algorithms, which does not explicitly appear in Algorithm 2.1, is the way the sub-problems are ordered in the search queue. A depth-first search using a Last In-First Out queue (a.k.a. a stack) can help find feasible solution quicker and maintain meaningful upper-bound, when no initial solution is available. An A^* search (best-first) based on a priority queue ordered by lower-bounds of the sub-problems is also a great alternative.

2.2.2 Constraint Programming

The other large class of methods to find exact solutions to VRPs and its variants is built around Constraint Programming (CP) [101, 91]. In CP, all variables are associated to their domains and linked together by the constraints in which they are involved. Constraint propagators update the domains of linked variables to avoid incompatible assignments. When all propagators have reduced the domains of the variables, a search branch can be expanded by committing to an assignment for a variable and propagating its consequences through constraints. If at some point this propagation leads to an empty variable domain, the candidate assignment is infeasible and the algorithm backtracks to the previous feasible state. Algorithm 2.2 depicts a recursive implementation of a basic search procedure for CP. The key points of the algorithm are: the way the constraints are propagated, and the order in which candidate assignment are tested.

```
1  function Solve(partialSolution)          /* empty solution for initial call */
2      PropagateConstraints(partialSolution)
3      if all domains are singletons then
4          return partialSolution
5      else if any domain is empty then
6          return failure
7      else
8          foreach candidateAssignement in partialSolution do
9              add candidateAssignement to partialSolution
10             if Solve(partialSolution) is solution then
11                 return partialSolution  /* and exit for loop                */
12             revert candidateAssignement  /* in case of failure               */
```

Algorithm 2.2: Constraint Programming

A classical way to model VRPs in CP is to define $n+2m$ successors variables $s_j \in [\![1, n+2m]\!]$ for every customer $j \in [\![1, n]\!]$ and as many copy of the depot as twice the number of vehicles m. These $2m$ special depot nodes represents vehicles leaving and returning to the depot. The "returning" node of every vehicle is constraint to be followed by the "leaving" node of the next vehicle (in a cycle), i.e. $s_{n+m+i} = s_{n+i+1}$ $\forall i \in [\![1, m-1]\!]$ and $s_{n+2m} = s_{n+1}$. This modelling enables the CP to use very powerful and efficient global propagators that involves

many variables and reduce all their domains at once, such as an "all different" and a "circuit" constraints on this successors variables. Adding new assignment variables $a_j \in [\![1, m]\!]$ for every customer $j \in [\![1, n]\!]$, the capacity constraint can be handled by a "bin-packing" subroutine and its specialized propagators.

If CP approaches are easier to adapt to new problem variants and enrich with more specific operational constraints, they are often slower than a well-defined ILP with an expertly designed Branch-and-Bound.

2.3 Heuristic methods for VRPs

After reviewing the two main classes of approach to find exact solutions to VRPs, we will now focus on approximation methods, which are the only one that can be applied in many real-case scenarios due to the computational complexity of VRPs. First, it is important to note that both ILP and CP methods presented as exact often maintain intermediate solutions which have not been proven optimal yet but are nonetheless valid approximations which can be retrieved at any time by interrupting the search. In this section, we will present search procedure which have no guarantee to converge to an optimal solution (and often no bounds on the relative performance of the solution compared to an optimal one), but are much quicker to find empirically good approximations in a majority of cases.

2.3.1 Tabu Search and Simulated Annealing

We start by reviewing these two variants of local neighborhood search that are Tabu Search [39] and Simulated Annealing [82]. Both rely on local operators that make minor modifications of an initial candidate solution to create new ones called its neighbors. If any neighbor yields better performances, it becomes the new candidate solution. However both algorithms allow worsening solution to become candidate in order to better explore the solution space and escape from local optima and plateaus. Their difference comes from this escape mechanisms: Tabu Search keeps a list of the last candidate solutions explored and prevents the search from coming back to it, while Simulated Annealing uses a random perturbation of solution in a neighborhood whose size shrink with time. For the same reason, they both allow a candidate solution to violate some constraints. They keep track of the last best valid solution (if any is ever found) which they can return at any time. Finally, they allow the user to interrupt the search based on any desirable stopping criterion. Algorithm 2.3 illustrates a combination of these two local search procedures.

```
1 candidate ← FindInitialSolution()
2 best ← candidate
3 while stopping criterion not met do
4     if IsValid(candidate) and Cost(candidate) < Cost(best) then
5         best ← candidate
6     neighbor ← FindBestNeighbor(candidate)
7     if Cost(neighbor) < Cost(candidate) or AllowWorse(neighbor) then
8         candidate ← neighbor
```

Algorithm 2.3: Local Search

Because of their very competitive running time for acceptable routes quality, Tabu Search and Simulated Annealing meta-heuristics have been widely used in combinations with local search heuristic such as Variable Neighborhood Search (VNS) to address Dynamic and Stochastic problems [38, 25, 95, 96]. However they blindly explore the solution-space and cannot exploit specific patterns in the customers configuration. It is non-trivial to identify such exploitable patterns, however it is possible to learn them from experiences, which is what we will contribute to in this thesis using Reinforcement Learning.

2.3.2 Approximate Dynamic Programming

Approximate Dynamic Programming (ADP) [83] addresses stochastic and dynamic VRPs from the point of view of a centralized dispatcher that takes decisions to allocate resources (vehicles) to tasks (deliveries). The state of the fleet is described as a very large integer vector that count the number of vehicles in any particular configuration (position, load, duty time, etc...). Effect of every decision is decomposed into a deterministic part (vehicles move to their assigned deliveries) and a stochastic part (lateness, unexpected load, etc...). Both parts update the state vector by decreasing and increasing the counters. To limit its size, one can built aggregated state vector where the configurations are quantized into larger bins.

This approach is naturally well-suited to address DS-VRPs, and is very close to our modelling approach which we will detail in a Chapter 5. Indeed it relies on a sequential decision model and propose concise state representation to compute sufficient statistics (value functions) updated recursively from successive decision steps. However it differs from our approach on the model of the problem and the approximation structures used to represent these statistics.

It has been successfully applied to large-scale truck routing [100] and train routing [16] problems. The model developed in [83] and used in both application case uses counters of vehicles and customers with certain characteristics to describe the state of the systems. The authors demonstrated the capacity of the ADP method to efficiently scale up to problems with up to thousands of vehicles. However, the counter-based state representation they use degenerates to one-hot vectors and exhaustive representation for smaller fleet where vehicles are considered individually. Hence, it cannot be easily applied to city logistics.

2.3.3 Evolutionary Algorithms

Evolutionary algorithms, in particular Genetic Algorithms (GAs) [68], have been successfully used to solve sub-problems in DS-VRP instances [64, 93]. From a population of initial candidate solutions, a new generation of solutions is generated by recombining and mutating the most promising solutions according to a fitness criterion. The process is iterated on many generations. The size of the population and the mutation encourage diversity in the exploration of solutions, while the fitness criterion guide this exploration towards the best solutions.

One of the advantage of GA is that they can easily integrate many other heuristics as recombination or mutation operators, hence indirectly benefiting from any progress made in the literature. This gives birth to many hybrid approaches which combine the efficient solution exploration offered by GAs and the performances of other heuristics, such as Large Neighborhood Search for example [93]. However, they cannot naturally anticipate on dynamic and stochastic information, except maybe internally by using lookup search and generative model while evaluating the fitness of each solution in the population.

2.3.4 Ant Colony Optimization

Ant Colony Optimization [101] is also a meta-heuristic that has proven to be useful to solve routing problems. As the name suggest, it is inspired by the pheromone mechanism used by ants to coordinates themselves. Each simulated ant travels through the edges of the graph to form a solution. At the end of the routes, it deposits pheromones on the edges it selected, proportionally to the quality of the solution it obtained based on the objective value. The process is then iterated, but ants select edges with higher probability if the pheromone trail is intense. Pheromones slowly evaporate after each iteration, such that only most promising edges remain at the end.

One interesting property of ACO in the case of DS-VRPs is that new problems arising from the evolutions of the situation (new customers, infeasibility of previous solution due to delays) can be solved using the pheromone trails obtained while solving the previous configuration of the system [72, 65]. This assume that the new information does not disrupt the current solution too much and that the old solution still contains interesting pattern which can biased the new search in the right direction. However, except for this interesting potential persistance of the pheromone, the utilization of ACOs in the dynamic and stochastic case often simply consists in restarting the search procedure in an updated, and frozen configuration of the customers and partial routes, without any anticipation of the future evolution of the system.

2.4 Focus on dynamic and stochastic problems

Now that we reviewed most of the major methods in the literature for routing problems, we will discuss a few recent work in the state-of-the-art focusing on dynamic and/or stochastic VRPs. Table 2.1 positions some references based on the methods they used and the problems variations they considered. It draws from the review of [90], and adds a few additional references.

	Problem flavors							*Var. properties*			
	Traveling Salesman Problem	Vehicle Routing Problem	Pickup and Delivery Problem	... with capacitated vehicles	... with electric vehicles	... with Time Window	... with Transfers	Dynamic replanning	Stochastic demand	Stochastic customers	Stochastic travel times
Chance Constrained Programming		[35, 55]		[35, 55]					[35]		[55]
Stochastic Programming with recourse		[55, 56]							[56]		[55]
Approximate Linear Programming	[110]								[110]		[110]
Approximate Dynamic Programming		[66, 74]	[100, 16]		[16][2]	[100, 16]	[16]	[74]	[66, 74]	[100, 16]	[100, 16]
Stochastic VNS		[96]	[95]	[95]		[95]		[95]		[95]	[95]
Multiple Plan Approach			[8, 95]	[8, 95]		[8, 95]		[8, 95]			
Multiple Scenario Approach			[8, 95]	[8, 95]		[8, 95]		[8, 95]		[8, 95]	[95]
Genetic Algorithm		[64]	[93]	[64, 93]		[93]		[64, 93]		[93]	[93]
Particle Swarm Optimization			[26, 18]	[26, 18]		[26]		[26]	[26]		
Auction-based algorithm			[67, 20]	[67, 20]		[67, 20]	[20]	[67, 20]			
Monte-Carlo Tree Search	[89]	[62]		[62]		[89]					

Table 2.1 – Selection and classification of resolution methods based on the variant of problem they were applied to.

[2]The work presented in [16] addresses a train routing problem with constrained "distance budget" before mandatory maintenance, which could be compared to the limited autonomy of electric vehicles.

Based on the references listed in this table, heuristic approaches seems to receive more focus in the last years than exact or approximate approaches to address rich DS-VRPs, because of the complexity of the problem. A variety of methods have been proposed, and all of them have demonstrated some success. The main take-away is that methods that can efficiently take into account and anticipate on stochastic and dynamic events through lookup strategies (e.g. Monte-Carlo trajectory sampling) have a net advantage in DS-VRPs compared to more reactive approach simply updating plans as soon as new information is available.

From this review, the Variable Neighborhood Search (VNS) introduced in [69] seems to have received a lot of attention in the last fifteen years. It can be seen as a specialized extension of Simulated Annealing where local-operators with increasing level of perturbation on the solution defines growing neighborhoods around the current solution. These neighborhoods are explored in sequence until a better solution is found. The VNS is allowed to go through invalid solutions by relaxing some of the constraints of the problem, but tries to return a valid one if it finds any. Its Stochastic variant S-VNS [43] uses a generative model to estimate the value of a solution on a short horizon. It was adapted to the Dial-a-Ride Problem (DARP) by [95]. The Multiple Plan Approach (MPA) and its stochastic variant the Multiple Scenario Approach (MSA), both introduced in [8], are approaches which maintain batches of solutions and revisit them based on dynamic events while executing consensual decisions. If the original paper used VNS to produce the candidate solutions in the batch, one could combine this framework with any other heuristic and meta-heuristic.

All these approaches still lack the ability to extract knowledge from past experiences, and blindly explore new solutions every time the customer configuration changes. That is why approaches based on Deep Neural Networks, and more specifically Deep RL, have been recently proposed to automatically learn heuristics based on data, as discussed in the following Section 2.5.2.

2.5 Deep Neural Networks Heuristics

The Deep Learning trend has reached many fields of Artificial Intelligence during the last twenty years, emphasized by some great successes in Image Recognition and Segmentation [87], Text Analysis and Translation [86], or Reinforcement Learning [71], to only cite a few. These techniques have recently reached the field of Intelligent Transport Systems [113]. Borrowing structures initially used for text encoding, and some training algorithms from Reinforcement Learning, a recent line of work has emerged to address some well-known combinatorial problems [7], including the Travelling Salesman Problem and the Vehicle Routing Problem. Deep Neural Networks can either be used as intermediate heuristics involved in more classical local search approaches, to evaluate a partial solution or select the next node to be inserted for example [47], or as an end-to-end planner that directly outputs a solution from an encoding of the raw characteristics of the problem instance [50]. They are capable of extracting patterns and knowledge from past experiences and historical data, learning compact and informative representation to build a solution for a new instance of the same problem without any human expert intervention. Combined with the fact that a forward pass computation can be highly parallelized, getting the best from GPU hardware acceleration, they have an incredible quality to speed ratio compared to all other existing heuristics. In this section, we will first review some basic background knowledge about Deep Learning, before detailing the recent advances in Combinatorial Optimization using Deep RL and Attention Mechanisms used as graph encoders.

2.5.1 Basics of Deep Learning

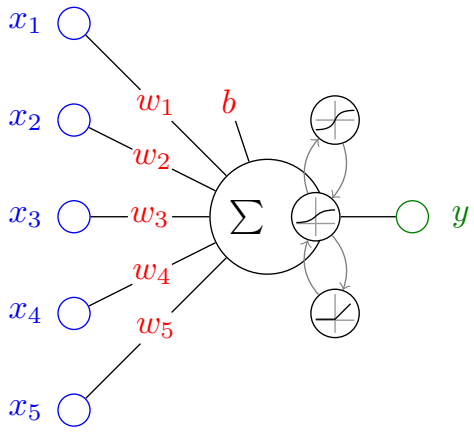


Figure 2.6 – Abstract representation of a neuron in a deep neural network. Its inputs (blue) are linearly combined, weighted by learned parameters (red). This combination is then fed to a non-linear activation function to form its output (green). Here, some alternatives for this activation function are represented: hyperbolic tangent (top), sigmoid (middle) or rectified linear unit (bottom).

Before diving at the heart of the problem, let's first introduce some basic notions of Deep Learning. A Deep Neural Network is a stack of layers composed of neurons. Each neuron computes a simple linear combination of its inputs, coming from previous layers of the network, which is then saturated by a non-linear *activation* function to yield this neuron's output, as illustrated on Figure 2.6. Theoretically[3], activation functions should be differentiable on $\mathbb{R}$. The linear combination is weighted by free parameters $\boldsymbol{w} \in \mathbb{R}^d$ which will be optimized during training, where $d \in \mathbb{N}$ is the size of the input and parameter vectors. It can be optionally offset by an additional *bias* parameter $b \in \mathbb{R}$.

On its own, a single neuron cannot learn much, but combined with *many* others, they are theoretically capable of learning *any* function. Layers of neurons are assembled as illustrated on Figure 2.7, forming a network called Multi-Layer Perceptron [92] (MLP).

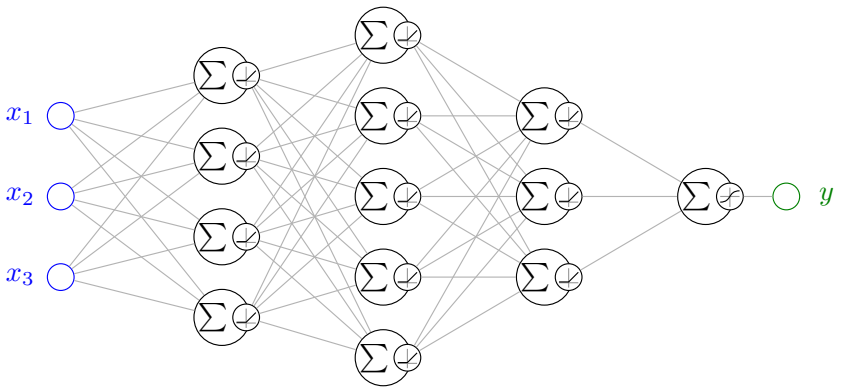


Figure 2.7 – A Multi-Layer Perceptron with three hidden layers of size 4, 5 and 3 and a single output neuron. This network could be trained as a simple binary classifier in an input space of dimension 3.

[3]In practice, the rectified linear unit and its variants are not differentiable at 0 but works anyway.

Mathematically, it is more practical to regroup every input, output, parameters and intermediate results as tensors. Let's consider for example an MLP of L layers of respective sizes $d_1, d_2, \ldots, d_L$. We note $\boldsymbol{x} \in \mathbb{R}^{d_0}$ the input vector, $\boldsymbol{y} \in \mathbb{R}^{d_L}$ the output vector. $\boldsymbol{w}_l \in \mathbb{R}^{d_{l-1} \times d_l}$ is the matrix of weights, and $\boldsymbol{b}_l \in \mathbb{R}^{d_l}$ the vector of biases of layer $l \in [\![1, L]\!]$. Any intermediate results $\boldsymbol{h}_l \in \mathbb{R}^{d_l}$ is computed as:

$$h_l = \Gamma_l(h_{l-1} \cdot w_l + b_l) \tag{2.6}$$

where Γ_l is the activation function of layer $l \in [\![1, L]\!]$ applied element-wise, $\boldsymbol{h}_0 = \boldsymbol{x}$ and $\boldsymbol{y} = \boldsymbol{h}_L$. The objective pursued while training the network is expressed as minimizing a scalar loss function ξ whose definition depends on the application. It at least takes the network output $\boldsymbol{y}$ as one of its input, and must be differentiable w.r.t. it, i.e. $\frac{\partial \xi}{\partial y_i}$ exists and is finite $\forall i \in [\![1, d_L]\!]$. Using the chain rule, this enables us to optimize the parameters of the network using first order optimization, a.k.a. gradient descent. Indeed for any layer $l \in [\![1, L]\!]$:

$$\frac{\partial \xi}{\partial \boldsymbol{w}_l} = \frac{\partial \xi}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{h}_{L-1}} \cdots \frac{\partial \boldsymbol{h}_{l+1}}{\partial \boldsymbol{h}_l} \frac{\partial \boldsymbol{h}_l}{\partial \boldsymbol{w}_l} \tag{2.7}$$

$$\boldsymbol{w}_l \leftarrow \boldsymbol{w}_l - \alpha \frac{\partial \xi}{\partial \boldsymbol{w}_l} \tag{2.8}$$

where (2.7) illustrates the chain rule, and (2.8) is the gradient descent update rule with $\alpha > 0$ the learning rate, a meta-parameter controlling the step-size of each update. Similar calculation can be written for the biases $\boldsymbol{b}_l$.

However, the objective is actually defined as minimizing the expected value of the loss w.r.t. some distribution of the input $\boldsymbol{X}$. The support of this distribution is often high-dimensional, so it is intractable to precisely compute the gradient. Instead, a dataset containing many samples of the input is fractioned into *mini-batches* of size B, for which we can efficiently compute corresponding mini-batches of output in what is called a *forward pass* into the network. The objective is estimated from the empirical mean of the loss function on this mini-batch, making the optimization algorithm a Stochastic Gradient Descent (SGD). The gradients w.r.t. the parameters of the different layers are also efficiently computed during a *backward pass* into the network. An *optimizer* then uses the estimated gradient values to update the parameters and the forward-backward passes are repeated until there are no more mini-batches and the whole dataset has been through the network. At this point, we have reached the end of a training *epoch*, some meta-parameters can be updated, and some accuracy and performance tests can be conducted using a separated dataset, containing samples never encountered during training to detect and avoid over-fitting. Afterwards, a new epoch starts using new mini-batches of the training dataset. The size of the mini-batches, the number of training epoch, the learning rate and the way it is updated at each epoch, the choice of optimizer, ... are as many meta-parameters we can tune to get the best performances from our network.

2.5.2 Using DNN to solve routing problems

Now that we are more familiar with some basic notions of Deep Learning, let us review how to train a network to solve routing problems. The goal is to learn to represent the customer configuration and the partial routes already travelled, and to output a distribution over the next step of the route. We will need architectures capable of efficiently representing structured data. Indeed, if images can directly be viewed as tensors, it is not the case for words in a text, or

nodes of a graph. We will not go into the details of how words are embedded and combined to encode a whole sentence. However we will see how the architectures developed for text can be used to encode fully-connected graphs such as the ones formed by customers in a VRP. As an illustrative example, we will consider the graph depicted on Figure 2.8, which corresponds to the toy DS-VRP presented earlier in Figure 2.5. There are 5 nodes representing the depot and the 4 customers, each of them labelled by a vector of attributes containing its position, demand, time window and duration. Because vehicles can travel from any node to any other node, the graph is fully-connected and all edges are labelled by vectors containing their travel costs and times.

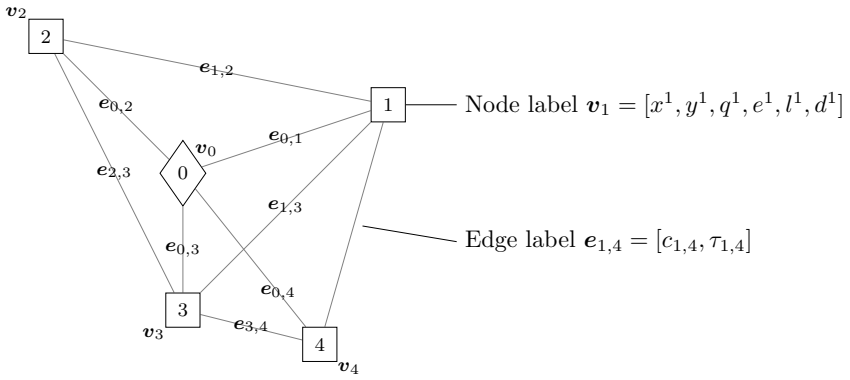


Figure 2.8 – Small graph of customers with vectors of attributes $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i,j}$ as labels on nodes and edges, respectively. Here, these vectors contains the customers features and travel costs and times of a VRPTW as an example. We need to encode the whole graph in order to use it as an input of a DNN.

The following of this section will describe state-of-the-art architectures to encode such a graph and to learn heuristics to solve VRPs on a distribution of instances with different configurations of customers. We will start with the pioneering approach of Vinyals, Fortunato, and Jaitly[116] which uses an advanced variant of Recurrent Neural Network (RNN) called Long Short-Term Memory (LSTM), and combines it with an additive attention mechanism to form an architecture called Pointer Network. We will then introduce a more recent architecture called Transformer [112] and its core Multi-Head Attention (MHA) layer based on multiplicative attention that it relies on. We will review its application to VRPs through the work of Kool, Hoof, and Welling [50].

In all this section, we will use the notation $\boldsymbol{w}_{\boldsymbol{y}}^{\boldsymbol{x}} \in \mathbb{R}^{\cdots \times d_x \times d_y}$ to indicate a tensor of learned parameters that projects an input vector $\boldsymbol{x} \in \mathbb{R}^{d_x}$ to a new vector denoted $\boldsymbol{y} \in \mathbb{R}^{d_y}$, where $d_x \in \mathbb{N}$ and $d_y \in \mathbb{N}$ are the input and output dimensions, respectively. If we need to index a component of such a tensor of parameters, we will use the notation $[\boldsymbol{w}_{\boldsymbol{y}}^{\boldsymbol{x}}]_i$ to help differentiate the name of the tensor from the index i of the component we want to select.

Recurrent Neural Networks

Before we dive into Pointer Networks, let us first introduce how a RNN accumulates information from a sequence of input vectors $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_n$ into a hidden state $\boldsymbol{h}$. We will denote $d_v \in \mathbb{N}$ the dimension of the input vectors and $d_h \in \mathbb{N}$ the dimension of the hidden state vector. At each input element $\boldsymbol{v}_j \in \mathbb{R}^{d_v}$ in the sequence, the hidden state $\boldsymbol{h}_j \in [-1, 1]^{d_h}$ gets updated as follow:

$$\boldsymbol{h}_{j+1} = \tanh(\boldsymbol{v}_j \cdot \boldsymbol{w}_{\boldsymbol{h}}^{\boldsymbol{v}} + \boldsymbol{h}_j \cdot \boldsymbol{w}_{\boldsymbol{h}}^{\boldsymbol{h}} + \boldsymbol{b}_{\boldsymbol{h}}) \tag{2.9}$$

where tanh is used as an activation function applied element-wise, $\boldsymbol{w}_{\boldsymbol{h}}^{\boldsymbol{v}} \in \mathbb{R}^{d_v \times d_h}$ is a matrix of parameters projecting the input element $\boldsymbol{v}_j$ to the hidden vector-space of the RNN, $\boldsymbol{w}_{\boldsymbol{h}}^{\boldsymbol{h}} \in \mathbb{R}^{d_h \times d_h}$ is a matrix of parameters re-projecting the previous hidden state $\boldsymbol{h}_j$ to the same hidden vector-space to be combined with the new input, and $\boldsymbol{b}_{\boldsymbol{h}} \in \mathbb{R}^{d_h}$ is an optional vector of biases. The initial hidden state $\boldsymbol{h}_0$ which initiate the output sequence can either be an arbitrary constant vector, or a learned parameter. To encode the example graph pictured in Figure 2.8 through a RNN, we would feed it with the nodes labels in any arbitrary order. With this architecture, we cannot take into account the edge labels (here: travel costs and times), and can only indirectly rely on the correlations between them and the nodes labels (e.g. customer coordinates). Figure 2.9 illustrates how the example graph previously introduced in Figure 2.8 would be encoded through a RNN.

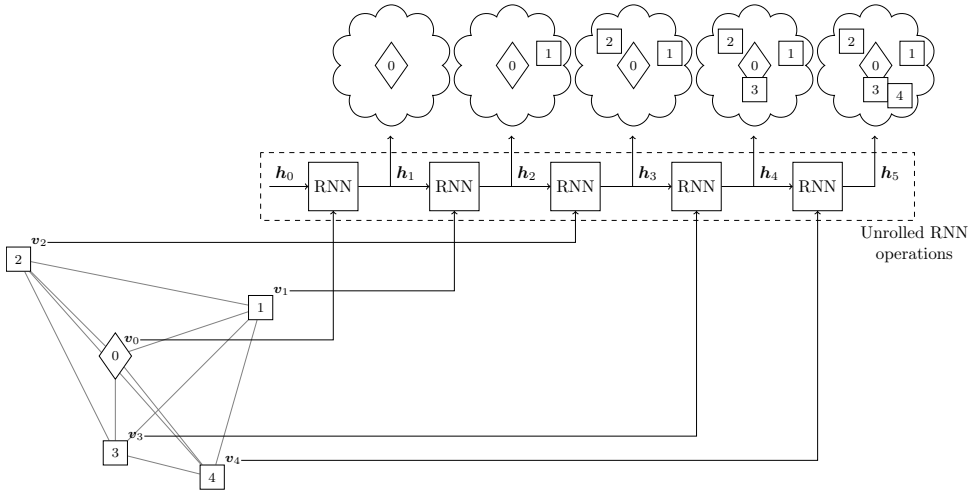


Figure 2.9 – Illustration of how a RNN can be used to encode the graph introduced in Figure 2.8. Each node label $\boldsymbol{v}_j$ is fed to the RNN in an arbitrary order to progressively build a representation $\boldsymbol{h}_5$ of the whole graph.

The interesting property of RNNs which make them useful for encoding graphs is that they can learn representation of sequences $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_n$ of variable lengths. Nonetheless, even though they re-use the same parameters for all input elements, they are not invariant to permutations in the order of these elements. This property greatly reduces their efficiency when trying to learn to represent an unordered set of nodes (through their labels). Additionally, RNNs accumulate information on prefixes of the sequence until they reach the last element $\boldsymbol{v}_n$, and their hidden states $\boldsymbol{h}_{n+1}$ finally contain a representation of the whole input. Because of this, we do not get a proper encoding for each node of the graph, as intermediate hidden states $\boldsymbol{h}_j$ lack the information contained in the suffix $\boldsymbol{v}_j, \ldots, \boldsymbol{v}_n$ of the sequence. The only representation that can really capture all the information available is the one obtained at the end, corresponding to the whole graph. Finally, as all elements are treated equally, RNNs have a tendency to forget older elements when the sequence length increases, because information gets diluted in the hidden state through successive updates.

To solve this last issue of RNNs, more advanced recurrent architectures have been proposed. For example, Long Short-Term Memory (LSTM) cells [46] add another internal vector called the cell state, denoted $\boldsymbol{c} \in \mathbb{R}^{d_h}$. This cell state can be seen as a read/write memory within the layer, and is updated through a set of four *gates* that learn what information should be forgotten, saved, or extracted from the cell state based on the successive input elements and hidden states. Although LSTMs try to counter-act the biases of RNNs that forget older inputs, they still suffer

from the same limitations when it comes to encoding the nodes of a graph. To solve the lack of dependencies between the intermediate hidden states and the suffixes of the input sequences, one could use bidirectional LSTM, where the reversed sequence is fed to another "backward" LSTM cell whose hidden states are then combined with the ones of the "forward" cell.

Pointer Networks

We have just presented how RNNs can be used to represent a graph, and we are now ready to review how to address the second challenge in applying RNNs to solving routing problems, which is to produce a permutation of the input sequence. In sequence-to-sequence (seq-to-seq) models, which are used to translate sentences from one language to another, two RNNs work in tandem. An encoder produces a hidden state for every element of the input sequence. Then, a decoder produces a sequence of output elements based on the context provided by the encoder. In the simplest seq-to-seq models, this context only consists in the final hidden state $\boldsymbol{h}_n$ of the encoder. To enrich the information available to the decoder, an additive attention layer $\boldsymbol{g}$ can be added, as describe as follow. It takes as input the whole sequence of encoder states $\boldsymbol{h}_1, \dots, \boldsymbol{h}_{n+1}$ regrouped as a tensor $\boldsymbol{h} \in \mathbb{R}^{n \times d_h}$ and the current hidden state of the decoder RNN $\boldsymbol{h'} \in \mathbb{R}^{d_h}$, and outputs a glimpse $\boldsymbol{g} \in \mathbb{R}^{d_h}$ which combines them as follow:

$$\boldsymbol{a} = \eta((\boldsymbol{h} \cdot \boldsymbol{w}_{\boldsymbol{a}}^{\boldsymbol{h}} + \overleftrightarrow{\boldsymbol{h}'} \cdot \boldsymbol{w}_{\boldsymbol{a}}^{\boldsymbol{h}'}) \cdot \boldsymbol{w}_{\boldsymbol{a}}) \tag{2.10}$$

$$\boldsymbol{g} = \boldsymbol{a} \cdot \boldsymbol{h} \tag{2.11}$$

where $\boldsymbol{w}_{\boldsymbol{a}}^{\boldsymbol{h}} \in \mathbb{R}^{d_h \times d_a}$, $\boldsymbol{w}_{\boldsymbol{a}}^{\boldsymbol{h}'} \in \mathbb{R}^{d_h \times d_a}$ and $\boldsymbol{w}_{\boldsymbol{a}} \in \mathbb{R}^{d_a}$ are matrices and vector of learned parameters, $\overleftrightarrow{\boldsymbol{h}'}$ denotes a matrix $\in \mathbb{R}^{n \times d_h}$ built from stacking n copies of the decoder state $\boldsymbol{h'}$, and η is the soft-max function. The vector $\boldsymbol{a} \in [0, 1]^n$ contains the attention scores associated with every encoder state. They weight a linear combination of the encoder states to form the enriched glimpse $\boldsymbol{g}$ driven by the current decoder state $\boldsymbol{h'}$.

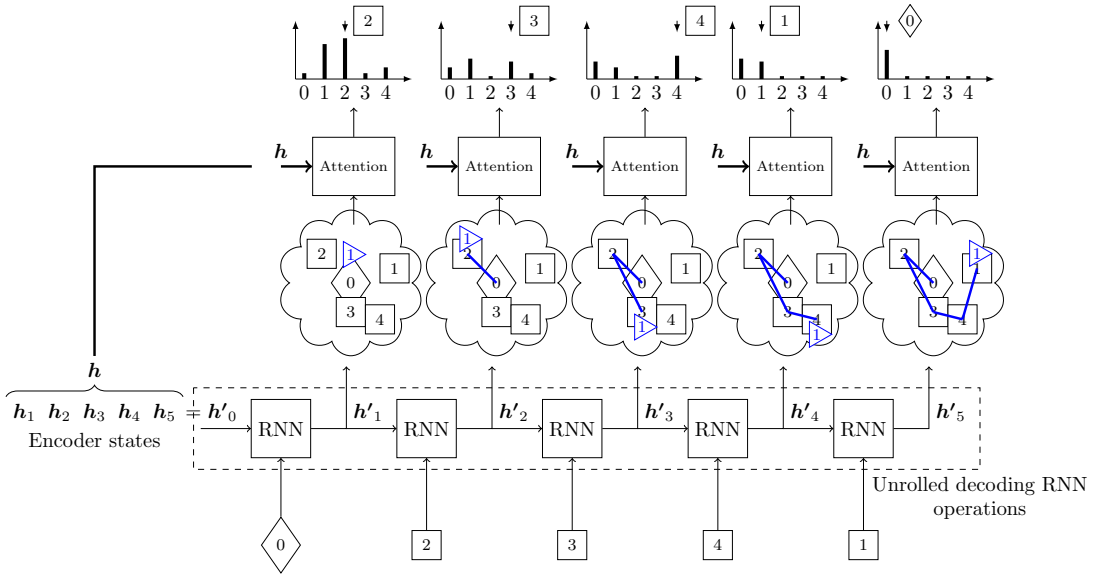


Figure 2.10 – Illustration of how a Pointer Network iteratively generates a permutation of the input nodes in the graph of Figure 2.8 to form a solution to the TSP.

In [116], Vinyals, Fortunato, and Jaitly proposed to use the same attention mechanism to output a permutation of the input sequence. Indeed the result of the Additive Attention glimpse

was previously used as an intermediate hidden state which was re-projected by further layers to form a distribution on an output dictionary completely independent of the input. Instead of using the attention vector $\boldsymbol{a}$ as weights to linearly combine the encoder hidden state and enrich the decoder state, they directly use it as the parameters of a distribution on the input sequence. This operation is illustrated on Figure 2.10. They sample this distribution to get the index of the next output selected from the input sequence, such that the probability to select a node $j \in [\![0, n]\!]$ as the next destination is equal to the attention score $\boldsymbol{a}_j$. In our example, this output index indicates which node to visit next.

They trained their model to solve 3 combinatorial problems on graphs: Convex Hull, Delaunay Triangulation, and more importantly for us TSP. For each problem, they randomly generated a million training instances, which they solved using existing exact or approximate algorithms to get ground truth labels. With this Supervised Learning approach, they could not outperform the heuristics they used to label the dataset.

In [6], Bello et al. extend the architecture by inserting some attention glimpses between the decoder and the final additive attention layer outputting the distribution on the input sequence. More importantly, they used a loss function borrowed from Reinforcement Learning to train their model directly from the performances of solutions sampled from the model. This self-improving approach enabled them to outperform the basic heuristics used as ground truth in [116]. However with no guidance from ground truth labels, there are many possibilities to explore in the solution space. The model even has to learn the "hard way" (i.e. by highly negative reward signals penalizing the performance measure) to avoid invalid solutions. However to temper this increased complexity, the authors added a masking mechanism applied on the distributions output by the model to prevent it from sampling invalid solutions.

These two first approaches used LSTM to encode the graph, and we already underlined that this structure does not have the desired property of invariance to permutations of its input. In [73], Nazari et al. propose to greatly simplify the initial encoding of the graph. They use a simple linear layer to project each node $\boldsymbol{v}_j \in \mathbb{R}^{d_v}$ to a intermediate representation $\boldsymbol{h}_j \in \mathbb{R}^{d_h}$. Formally, this translates to:

$$\boldsymbol{h}_j = \boldsymbol{v}_j \cdot \boldsymbol{w}_{\boldsymbol{h}}^{\boldsymbol{v}} + \boldsymbol{b}_j \tag{2.12}$$

where $\boldsymbol{w}_{\boldsymbol{h}}^{\boldsymbol{v}} \in \mathbb{R}^{d_v \times d_h}$ and $\boldsymbol{b}_j \in \mathbb{R}^{d_h}$ are a matrix and a vector of learned parameters. They use this flexibility and efficiency to address the CVRP, by re-encoding customers when they get served to update the internal representation of the state of the problem. However, because there is no relation between the encoding of each node, this efficiency comes with a cost in the complexity of the patterns the model can extract from the graphs of customers.

Multi-Head Attention

We now want to get rid of the RNN structure in favor of a permutation-invariant structure. To this end, the Multi-Head Attention (MHA) layer is a promising building block for efficient graph encoders. In a recent article [112], Vaswani et al. introduced this MHA layer for structured-data representation, originally applied to Natural Language Processing. We mentioned earlier that the additive attention in the existing structures (i.e. Pointer Networks and variants) played an important role in aggregating and focusing information from all nodes of the graph. Here, they propose a Multiplicative Attention model which can be highly parallelized, and defines attention scores based on a learned similarity measure. Moreover, their proposition is to split the computation to multiple heads, which offer more diversity in the internal intermediate representation. Thereby they obtain what is now called a Multi-Head Attention (MHA) layer.

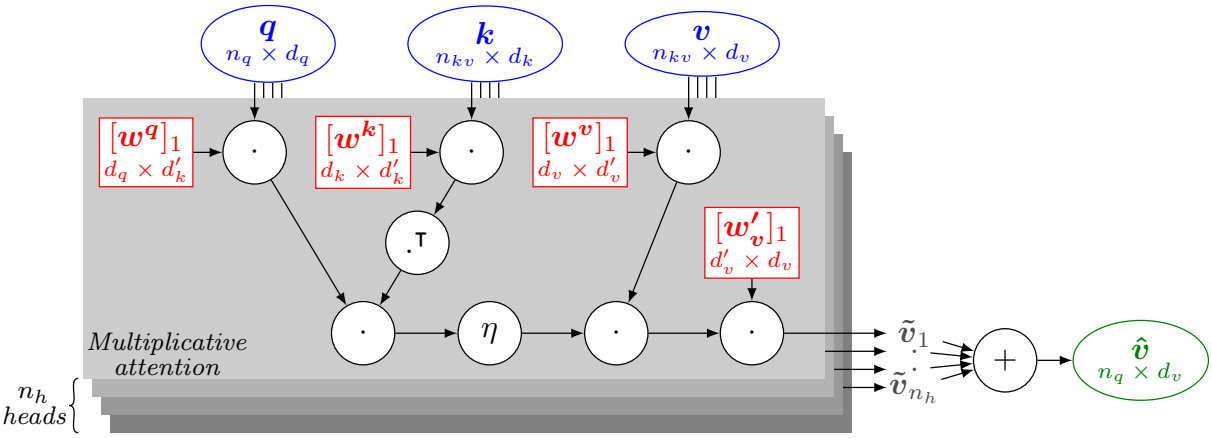


Figure 2.11 – Multi-Head Attention layer which combines a set of values regrouped in tensor $\boldsymbol{v}$ based on the similarity between their associated keys $\boldsymbol{k}$ and some queries $\boldsymbol{q}$. Different combinations $\tilde{\boldsymbol{v}}_h$ are produced in multiple parallel heads, and finally recombined into an output value $\hat{\boldsymbol{v}}$ for every query.

As illustrated in Figure 2.11, it takes as input a set of n_q queries of size d_q, and a set of n_{kv} key-value pairs of respective sizes d_k and d_v. Splitting the computation onto n_h parallel heads, it aggregates the values weighted by compatibility scores between their respective keys and the queries. Mathematically, for every head h, this compatibility $\langle \boldsymbol{q}, \boldsymbol{k} \rangle_h$ between a matrix of queries $\boldsymbol{q} \in \mathbb{R}^{n_q \times d_q}$ and a matrix of keys $\boldsymbol{k} \in \mathbb{R}^{n_{kv} \times d_k}$ can be written as:

$$\langle \boldsymbol{q}, \boldsymbol{k} \rangle_h = (\boldsymbol{q} \cdot [\boldsymbol{w}^{\boldsymbol{q}}]_h) \cdot (\boldsymbol{k} \cdot [\boldsymbol{w}^{\boldsymbol{k}}]_h)^\intercal \tag{2.13}$$

where $[\boldsymbol{w}^{\boldsymbol{q}}]_h \in \mathbb{R}^{d_q \times d'_k}$ and $[\boldsymbol{w}^{\boldsymbol{k}}]_h \in \mathbb{R}^{d_k \times d'_k}$ are matrices of learned parameters, specific to each head $h \in [\![1, H]\!]$, that projects $\boldsymbol{q}$ and $\boldsymbol{k}$ in an intermediate space of size d'_k where their scalar product plays the role of a similarity measure. Again, the notation $[\boldsymbol{w}^{\boldsymbol{q}}]_h$ indicates the component of the tensor $\boldsymbol{w}^{\boldsymbol{q}}$ containing the query projection weights of the h^{th} head of the MHA layer. Then, each head $h \in [\![1, H]\!]$ computes a value per query as a matrix $\tilde{\boldsymbol{v}}_h \in \mathbb{R}^{n_q \times d'_v}$:

$$\tilde{\boldsymbol{v}}_h = \eta\left(\frac{\langle \boldsymbol{q}, \boldsymbol{k} \rangle_h}{\sqrt{d'_k}}\right) \cdot (\boldsymbol{v}.[\boldsymbol{w}^{\boldsymbol{v}}]_h) \tag{2.14}$$

where $[\boldsymbol{w}^{\boldsymbol{v}}]_h \in \mathbb{R}^{d_v \times d'_v}$ is another matrix of learned parameters. In this intermediate representation, the contribution of every input value in $\boldsymbol{v}$ is weighted by the normalized similarity measure between its corresponding key in $\boldsymbol{k}$ and each query in q. Finally all head values are recombine through:

$$\hat{\boldsymbol{v}} = \sum_{h=1}^{H} \tilde{\boldsymbol{v}}_h.[\boldsymbol{w}'_{\boldsymbol{v}}]_h \tag{2.15}$$

where $\{[\boldsymbol{w}'_{\boldsymbol{v}}]_h \in \mathbb{R}^{d'_v \times d_v} \ \forall h\}$ are the last matrices of learned parameters that project every head values back to a single value per query $\hat{\boldsymbol{v}} \in \mathbb{R}^{n_q \times d_v}$ in the original value space.

This MHA layer can easily be used to aggregate information in a fully-connected graph, and has the advantage of producing a unique representation for each node (contrary to RNNs in [116]) but also encode the relations between the nodes (contrary to independent linear projection with shared parameters in [73]). Another net advantage of a MHA layer compared to a RNN cell is that all computations can be conducted in parallel, whereas successive hidden state can only be computed sequentially. Figure 2.12 illustrates how a MHA layer could be used to

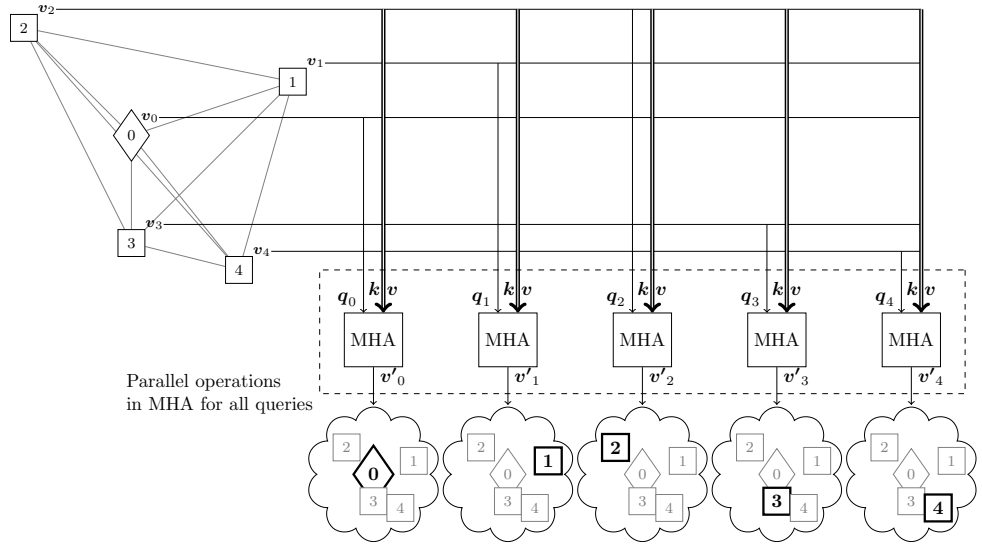


Figure 2.12 – Illustration of how the representations for every node in the graph of Figure 2.8 are obtained using a MHA layer.

encode the graph presented in Figure 2.8. The nodes attributes $\boldsymbol{v} \in \mathbb{R}^{n \times d_v}$ are fed to all three inputs of the layer, used as a *self-attention* mechanism. For any head $h \in [\![1, H]\!]$, the compatibility scores $\langle \boldsymbol{v}, \boldsymbol{v} \rangle_h \in \mathbb{R}^{n \times n}$ can be viewed as a scalar edge attribute between any pair of nodes. If the graph is not fully connected, the scores can be masked by forcing some elements to $-\infty$, thereby preventing the propagation of information between the corresponding nodes. However, if the graph is too sparse, a lot of useless computation will be conducted, and the MHA layer could be replaced by a more efficient structure. Finally, MHAs as graph encoders cannot naturally integrate edge labels into their computations, hence their application to VRPs still rely on features only associated with the customers, e.g. travel times and costs correlated to the customers coordinates.

Transformers

If the MHA layer could be used as a graph encoder on its own, it would not form a really deep architecture and would lack the capacity to represent richer non-linear functions. In [112], the authors then integrated the MHA layer into a larger structure called a Transformer, designed to enrich the intermediate representations by adding non-linearities in-between self-attention passes. This Transformer can then be used as a more powerful and expressive graph encoder. An illustration of the architecture of a Transformer is given in Figure 2.13. After getting an initial embedding $\boldsymbol{h}_0 \in \mathbb{R}^{n \times d_h}$ of the nodes attributes $\boldsymbol{v} \in \mathbb{R}^{n+1 \times d_v}$, successive Transformer layers first apply an MHA sub-layer used as a self-attention mechanism It combines and updates the attributes of the nodes by pulling information from all others. Through this learned aggregation and update function, it can extract more complex patterns formed by any subset of nodes in the graph. It is followed by a simple feed-forward layer (2-layers MLP) that adds non-linear relations to enrich the class of functions the Transformer can learn.

Both sub-layers (MHA and feed-forward) are short-circuited by skip-connections to improve gradient flow, and normalized to avoid biases due to the number of nodes. During training, the normalization operation keeps running averages of the mean and standard deviation of the intermediate attributes of the nodes. These estimations are saved along the other parameters

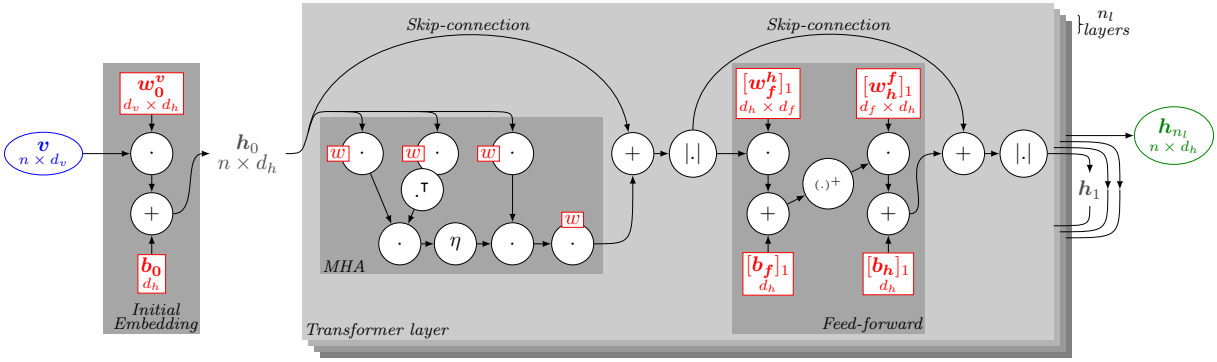


Figure 2.13 – Transformer Encoder. The structure is not recurrent and the n_l layers do not share the same weights. The new notations that are introduced are $|.|$ representing a normalization operation and $(.)^+$ a rectified linear unit (relu).

of the model, and their values can be frozen at inference time. Some dropout can be inserted to improve robustness and avoid over-fitting. This consists in randomly setting some elements of a tensor to 0 with a fixed probability, and emphasizes redundancy and diversity in the intermediate projections learned during training.

Attention Model

Deudon et al. [31] integrate the Transformer Encoder of [112] into an architecture designed to sample high-quality initial solutions for TSPs, which are then simply refined by a `2-opt` heuristic. In parallel to this work, Kool, Hoof, and Welling [50] also build around the Transformer Encoder and the Multi-Head Attention layer to propose an end-to-end heuristic that can efficiently sample solutions for a variety of problems ranging from VRPs to Stochastic TSPs. Additionally to this architecture contribution, they introduce a Monte-Carlo baseline to reduce the variance of the gradient estimator during training.

Their Attention Model (AM) learn to represent the graph of customers through a Transformer encoder. From the encodings of the customers and depot, they create a context vector which gets updated after each decision. This context vector drives an attention mechanism which computes a score for each node, used as the probability to select it as next target. The MARDAM architecture we built to address rich DS-VRPs we will present in Chapter 5 is inspired by and can be seen as an extension of AM. It uses the same attention mechanisms, namely Transformer and MHA, to encode the global state of customers and vehicles and build individual decision rules for every agent controlling the vehicles. We will also use AM in Chapter 6 as a state-of-the-art baseline for solving VRPs using Deep RL in the experimental benchmarks we use to evaluate the performances of MARDAM.

2.6 Conclusion

Vehicle Routing Problems (VRPs) have come a long way since their introduction in the late fifties [28]. Exact approaches such as the modern Branch-and-Cut-and-Price of [79] have found provably optimal solutions to larger and larger instances of the static and deterministic VRP. However the problem is still quite involving, and instances dealing with more than a few hundreds customers can only get approximate solutions, where progress is made towards reducing the

optimality gap. Other heuristics and meta-heuristics such as Genetic Algorithm, Ant Colony Optimization or Variable Neighborhood Search have been developed and are used extensively to address larger VRPs with a collection of additional constraints. If they do not provide a way to measure the optimality gap of the solution they return, it has been experimentally observed that they tend to produce high quality solutions. These heuristics are also more easily integrated into frameworks dealing with dynamic and stochastic information. The Multiple Plan Approach and Multiple Scenario Approach of [8] is a typical example of such frameworks. Nonetheless, most of these approximate approaches rely on hand-crafted expert heuristics and do not automatically learn from past experiences. They blindly explore the solution space given the customer configuration they are facing. One notable exception in the OR literature is the work of [83] on Approximate Dynamic Programming which exploit historical data from large logistic networks (e.g. long-haul trucks [100], or railroads [16]) to learn dispatching policies indicating how to allocate vehicles considered as resources to customer services or tasks.

A recent line of works initiated by [116] tried to use Deep Neural Network to learn to solve routing problems directly from data. Following works [6, 73] improved on the method and used Deep Reinforcement Learning to learn efficient heuristic for the Travelling Salesman Problem (TSP) and VRP without supervision. The complexity in applying such methods is to find an efficient way to represent the graph formed by customers and the state of the system. The latest improvement to address this challenge that has been proposed in the literature is Attention Mechanism, especially the Multi-Head Attention layer and the Transformer encoder of [112]. They have been successfully used to solve TSPs and VRPs with a few variants in [50]. However, all existing approaches based on Deep RL have reframed the VRP as a single-vehicle problem, and do not take into account the multi-agent dimension of the problem. Hence, they cannot represent the global state of the system in a dynamic and stochastic environment, where multiple vehicles evolve in parallel. In the following of this thesis, we will study such a multi-agent model and address it using an approach based on DNN, more specifically MHA and Transformer layers.

Chapter 3

Multi-Agent Reinforcement Learning

We just reviewed in Chapter 2 traditional models and solution methods for the Vehicle Routing Problem (VRP). Its basic static and deterministic variant is already computationally involving, and considering dynamic and stochastic events only makes the complexity of the problem grow. To avoid having to consider decision variables for every possible realizations of the stochastic variables, and every possible evolution of the dynamic ones upfront, most traditional approaches in the OR literature for the Dynamic and Stochastic VRP (DS-VRP) tend to solve new problems with freezed variables every time new information is available.

Markov Decision Processes offer a general framework that naturally describes agents interacting with a stochastic environment. If the dynamic state representation of DS-VRPs will still be a challenge in this MDP framework, it opens up opportunities to use all the recent tools of Deep Multi-Agent Reinforcement Learning (Deep MARL) which have demonstrated their capabilities to represent and compress complex state space while empirically performing near optimally. In this chapter, we will first introduce the formalization and some basic properties of Markov Decision Processes in Section 3.1 their direct extension to multi-agent (MMDP) settings in Section 3.1.2, and their decentralized, partially observable (Dec-POMDP) variant in Section 3.1.4. Then we will review the main approaches to optimize policies that provide decision rules to the agent while it interacts with its environment in Section 3.2. We will then report our preliminary work that contributed to extend the mathematical foundations of Policy Gradient methods, especially the Actor-Critic algorithms, to the decentralized setup in Section 3.3. This work was published in [2].

3.1 Markov Decision Processes

In this first section, we are going to present the different models we are interested in, by progressively introducing the properties we want to capture, namely the multi-agent dimension and the partial observability of the state. We will also present the more complex Dec-POMDP setting, where agents must take their decisions independently, based on their own individual observations. We will highlight how each of this model can be reduced to the simple MDP case using complex, and often continuous, augmented state spaces. This help understand how all the mathematical properties of MDPs can be generalized to more complex variants. It also highlights the challenge of finding efficient and rich state representations for this augmented state spaces, so that we can adapt the algorithms and resolution methods to the more complex cases.

3.1.1 Basics of MDPs

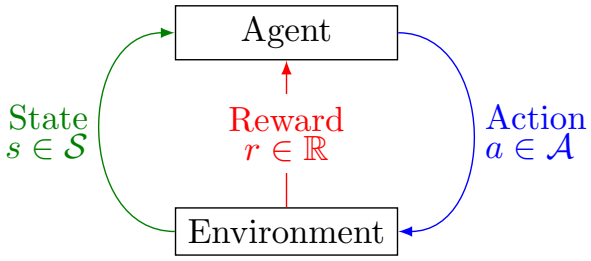


Figure 3.1 – Simple representation of the interaction loop modelled by an MDP. At each decision step, the agent observes the state (green arrow) and executes an action (blue arrow) to gather rewards (red arrow).

In a Markov Decision Process (MDP), an agent controls the evolution of an environment, in an interaction loop illustrated on Figure 3.1. After observing the environment state, the agent makes a decision and executes an action, which affects the environment and makes it transition to a new state. State transitions are generally stochastic, and obeys the Markov property which imposes that every transition only depends on the previous state and the last action. The agent also receives a reward after each transition, and will try to choose its action to accumulate as much reward as possible. Formally, an MDP is defined as follow:

Definition 3.1 (Markov Decision Process)**.** An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, P_0)$ where:

$\mathcal{S}$ is the set of possible states of the system;
$\mathcal{A}$ is the set of possible actions to interact with the system;
$P : (s, a, s') \mapsto \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\}$
is the transition rule which gives the probability of reaching state $s' \in \mathcal{S}$ after executing action $a \in \mathcal{A}$ from state $s \in \mathcal{S}$ – we will use a shorter mixed notation and write $P(s'|s, a)$;
$R : (s, a, s') \mapsto R(s, a, s') \in \mathbb{R}$
is the immediate reward function;
$P_0 : s \mapsto \Pr\{S_0 = s\}$
is the initial state distribution.

Depending on the system we want to model as an MDP, we can consider a few different variant to define how long the agent will interact with its environment. We call this number of decision steps the planning horizon, denoted T. It can either be finite ($T \in \mathbb{N}^+$), infinite ($T = \infty$) or undetermined. This last case corresponds to models where specific termination conditions defines a subset $\mathcal{G}$ of goal states that put an end to the episode once reached, after a finite but unknown number of decision steps. In many cases, there are more than one way to model a system in term of planning horizon. It is even possible to convert one horizon model to another with only minor modification to the objective function, that yields the same results. The fundamental properties on MDP are mostly the same whatever the nature of the planning horizon is. As an example, we could translate a finite horizon model to an infinite horizon one by augmenting the state with a step counter t and define a sink state that repeats indefinitely with 0 reward once $t \geqslant T$. With that in mind, let's now formally define how our agent takes its decision through what we call its policy, denoted π:

Definition 3.2 (Policy)**.** A non-stationary stochastic policy π provides decision rules $\pi_t : (s, a) \mapsto \Pr\{A_t = a \mid S_t = s\}$ which give the probability of choosing action $a \in \mathcal{A}$ from any state $s \in \mathcal{S}$ at any decision step $t \in [\![0, T]\!]$. We will use the short-hand notation $\pi_t(a|s)$.

For infinite horizon settings, we consider stationary policies π that do not vary in time, and prescribe actions $a \in \mathcal{A}$ given a state $s \in \mathcal{S}$. Policies can also be deterministic functions of the form $\pi : \mathcal{S} \to \mathcal{A}$, which can be seen as a degenerated conditional distributions that associate a specific action a to any state s with probability 1. We will see later that there always exists an optimal deterministic policy. However, the more general stochastic ones have the advantage of being continuous, which can make them easier to represent with some policy approximation architecture, such as deep neural networks. Stochastic policies can also be differentiable, which opens up opportunities to train them using gradient descent algorithms.

The objective in an MDP is to find a policy that maximizes the expected cumulated reward

$$\mathrm{E}_{\pi,P,P_0}\left[\sum_{t=0}^{T-1} \gamma^t R(S_t, A_t, S_{t+1})\right] \tag{3.1}$$

where $\gamma \in]0, 1]$ is a discount factor balancing the objective between immediate and future rewards, $S_0 \sim P_0$ and $A_t \sim \pi_t(\cdot|S_t)$ $S_{t+1} \sim P(\cdot|S_t, A_t)$ $\forall t \in [\![0, T-1]\!]$. If the planning horizon is infinite ($T = \infty$), we need $\gamma < 1$, otherwise the problem is undecidable.

3.1.2 Multi-agent MDPs

The VRPs which we want to solve using Reinforcement Learning (RL) involve a fleet of multiple vehicles. Hence, we need models that can capture the interactions of multiple agents with a shared environment. One of these models derives from MDP and is called Multi-agent MDP (MMDP) [15]. In MMDP, the evolution of the state and the rewards all agents receive result from the effect of all their individual actions combined into a joint action. They share a common objective which is to cooperate to maximize the expected cumulated reward. Given Definition 3.1 of an MDP, an MMDP is defined as follow:

Definition 3.3 (Multi-agent MDP)**.** An MMDP is defined by the tuple $(\mathcal{I}, \mathcal{S}, \mathcal{A}, P, R, P_0)$ where $\mathcal{S}, P, R, P_0$ have the same definitions as in MDPs and:

$\mathcal{I}$ is the set of agents interacting in the system;
$\mathcal{A} = \bigtimes_{i \in \mathcal{I}} \mathcal{A}^i$
is the set of possible joint actions which can be decomposed into individual actions $a^i \in \mathcal{A}^i$ chosen independently by every agent $i \in \mathcal{I}$;

3.1.3 Partially Observable MDPs

Unfortunately, some systems require way to many variables to describe their state evolution from one step to the next, and the agent is often forced to base its decisions on noisy and limited measurements correlated to this inaccessible and sometimes intractable environment state. This new model is called Partially Observable MDP (POMDP). This has for consequences that we

can no longer devise policies based on the state, which had the advantage of summarizing all the previous steps due to the Markov property. To guarantee that we do not loose any information, we a-priori need to keep track of all past observations and actions taken, which we call the history of the agent. Figure 3.2 illustrates the new interaction loop between the agent and the partially observable environment it evolves in.

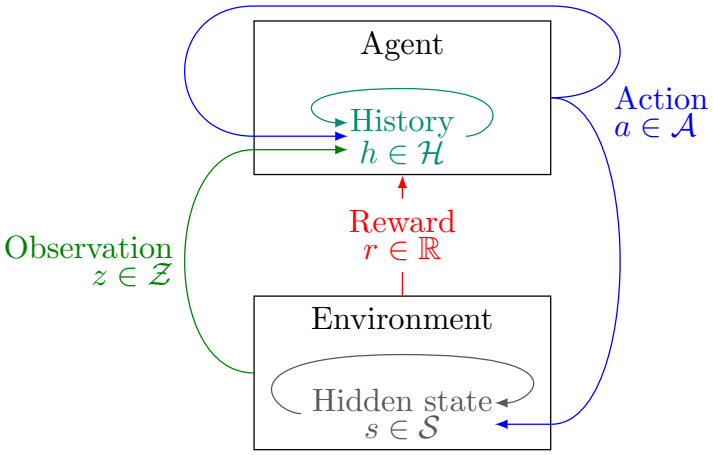


Figure 3.2 – Interaction loop between an agent and its partially observable environment. The state updates (gray arrow) are hidden, and only perceived through an imperfect observation (green arrow). The agent keeps track of all past observations and actions taken that constitute its history (cyan arrow), to accumulate information and drive its decisions to choose an action (blue arrow). It still receives rewards (red) and tries to accumulate as much as possible along its trajectory.

Definition 3.4 (Partially Observable MDP)**.** A Partially Observable Markov Decision Process (POMDP) is a variant of MDPs defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, P, \Omega, R, P_0)$ where $\mathcal{S}, \mathcal{A}, P, R, P_0$ have the same definitions as in MDPs and:

- $\mathcal{Z}$ is the set of possible observations received from the system;
- $\Omega : (s, z) \mapsto \Pr\{Z_t = z \mid S_t = s\}$ is the observation rule which gives the probability of receiving observation $z \in \mathcal{Z}$ from state $s \in \mathcal{S}$ – we will use a shorter mixed notation and write $\Omega(z|s)$;

Definition 3.5 (History)**.** An history $h_t \in \mathcal{H}_t$ aggregates all the information available to the agent up to decision step $t \in [\![0, T-1]\!]$. h_0 starts as a single-element sequence containing the initial observation $z_0 \in \mathcal{Z}$, and is recursively updated at each decision step t to include the last action $a_{t-1} \in \mathcal{A}$ and the resulting observation $z_t \in \mathcal{Z}$. Formally, we can write:

$$h_0 = (z_0,) \tag{3.2}$$

$$h_t = h_{t-1} \mathbin{+\!\!+} (a_{t-1}, z_t) = (z_0, a_0, \ldots, a_{t-1}, z_t) \qquad \forall t \in [\![1, T-1]\!] \tag{3.3}$$

where $+\!\!+$ is the concatenation operator.

$\mathcal{H}_t = \mathcal{H}_{t-1} \times \mathcal{A} \times \mathcal{Z}$ is the set of possible histories at decision step $t \in [\![1, T-1]\!]$ (with $\mathcal{H}_0 = \mathcal{Z}$), and we call $\mathcal{H} = \bigcup_{t=0}^{T-1} \mathcal{H}_t$ the set of all possible histories.

Definition 3.6 (Policy)**.** A non-stationary policy π provides decision rules $\pi_t : (h, a) \mapsto \Pr\{A_t = a \mid H_t = h\}$ for each step $t \in [\![0, T-1]\!]$ which give the probability of choosing action $a \in \mathcal{A}$ after history $h \in \mathcal{H}$. We will use the short-hand notation $\pi_t(a|h)$.

The major drawback of this partially observable model is that the cardinality of the set $\mathcal{H}$ of all possible histories grows exponentially with the planning horizon T. It is then mandatory to find a way to compress it, through concise sufficient statistics that preserve the information required to take decisions. Fortunately, we can prove that there exists such a statistic which is called belief state, and is formally defined as follow:

Definition 3.7 (Beliefs)**.** A belief state b_t is a statistic summarizing histories which indicate the probability to reach any state s after t steps. It is defined as follow:

$$b_t : s \mapsto \Pr\{S_t = s\} \qquad \forall t \in [\![0, T-1]\!] \tag{3.4}$$

If the transition model is known, belief can be efficiently updated from step to step given the previous belief state, the last action taken and the observation received. The following recursive update rule results from the Markov property and the Bayes rule.

Theorem 3.1 (Beliefs recursion)**.** *Knowing the previous belief state b_{t-1}, after taking action $a \in \mathcal{A}$ and observing $z \in \mathcal{Z}$, the new belief state b_t can be deduced through the following recursive relation:*

$$b_t(s') = \frac{1}{\Pr\{z'|b_{t-1}, a\}} \Omega(z'|s') \sum_{s \in \mathcal{S}} P(s'|s, a) b_{t-1}(s) \qquad \forall s' \in \mathcal{S} \qquad \forall t \in [\![1, T-1]\!] \tag{3.5}$$

and initially:

$$b_0(s) = P_0(s) \qquad \forall s \in \mathcal{S} \tag{3.6}$$

where $\Pr\{z'|b_{t-1}, a\} = \sum_{s' \in \mathcal{S}} \Omega(z'|s') \sum_{s \in \mathcal{S}} P(s'|s, a) b_{t-1}(s)$ in the normalization factor can be computed by marginalizing over states.

Without the transition model, one can try to maintain approximate beliefs through sampling and/or model learning. There are however many alternatives to construct approximate representations that drive decision rules, which we call internal state of the agent, e.g. truncated histories or recurrent neural networks. The belief states and the update rule defined in Theorem 3.1 can be used to define an augmented MDP where they respectively play the roles of states and transition function.

3.1.4 Decentralized POMDPs

Finally, we want a model that can capture the properties of systems where multiple agents control the environment but must take their decision in a decentralized way, based on their own individual observations. This model is called Decentralized POMDP (Dec-POMDP), and is much harder than all the variants we introduced before. All agents perceive the environment through their own incomplete and imperfect measurements, and cannot communicate with each other, at least not losslessly or not instantaneously. The hidden state of the environment evolves under the influence of all agents actions combined, and correlates the individual observations. The reward signal is shared and all agents must cooperate to maximize its accumulation along time. Figure 3.3 illustrates the interaction loop between 3 agents and their environment.

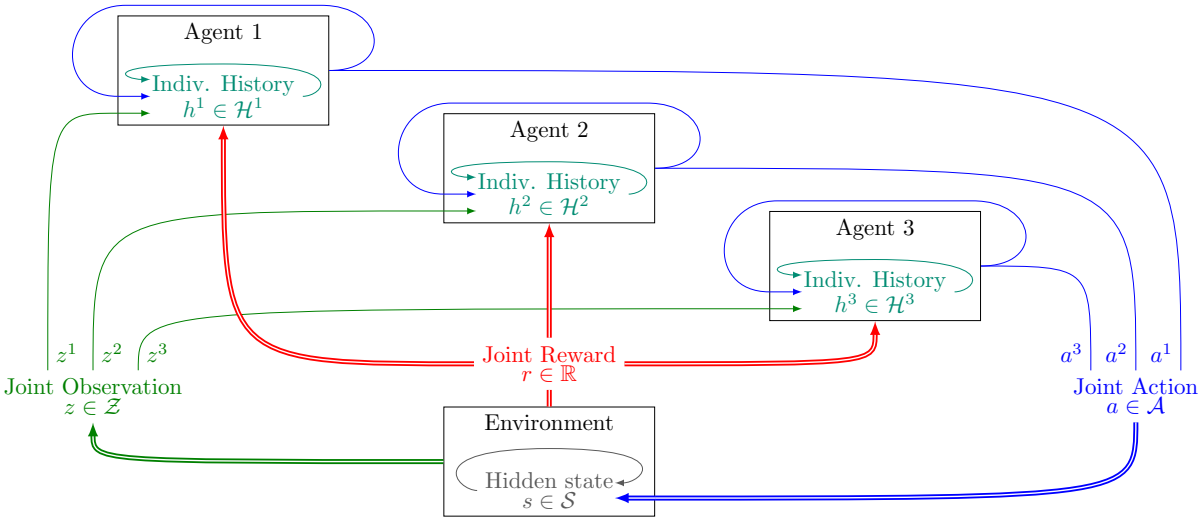


Figure 3.3 – Interaction loop between 3 agents and their environment. Agents receive correlated individual observations (green arrows) which they cannot share with each other. They choose their individual actions separately based on the information they have available stored in individual histories (cyan arrows). The combined effect of their action (blue arrows) makes the environment transition to a new state (gray arrow) and emit a joint reward signal (red arrows) which all agents tries to maximize, enforcing their cooperation and coordination. Notice that **bold** arrows represent joint or common variables, while normal arrows are the individual components.

Given Definition 3.3 of an MMDP and Definition 3.4 of a POMDP, Dec-POMDPs are defined as follow:

Definition 3.8 (Decentralized POMDP)**.** A Decentralized Partially Observable Markov Decision Process (Dec-POMDP) is defined by the tuple $(\mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{Z}, P, \Omega, R, P_0)$ where $\mathcal{I}, \mathcal{S}, \mathcal{A} = \times_{i \in \mathcal{I}} \mathcal{A}^i, P, R, P_0$ have the same definitions as in MMDPs, Ω is the joint observation function as defined in POMDP and:

$$\mathcal{Z} = \bigtimes_{i \in \mathcal{I}} \mathcal{Z}^i$$

is the set of possible joint observations that are composed of individual observations $z^i \in \mathcal{Z}^i$, each received only by a specific agent $i \in \mathcal{I}$;

Similarly to what we introduced for a single agent in POMDP, we will formalize the information available to each individual agent $i \in \mathcal{I}$ stored as individual histories which are concatenations of past individual observations and actions.

Definition 3.9 (Individual history)**.** An individual history $h_t^i \in \mathcal{H}_t^i$ aggregates all the information available to a specific agent $i \in \mathcal{I}$ at decision step $t \in [\![0, T-1]\!]$. It starts as a single-element sequence containing the initial individual observation $z_0^i \in \mathcal{Z}^i$, and is recursively updated at each decision step t to include the last individual action $a_{t-1}^i \in \mathcal{A}^i$ and the resulting individual observation $z_t^i \in \mathcal{Z}^i$ (see Definition 3.5 for a more formal description). We can regroup individual histories $\{h_t^i \ \forall i \in \mathcal{I}\}$ into a joint history $h \in \mathcal{H}_t$.

Definition 3.10 (Individual policy)**.** An non-stationary individual policy π^i provides decision rules $pi_t^i : (h^i, a^i) \mapsto \Pr\{A_t^i = a^i \mid H_t^i = h^i\}$ to a specific agent $i \in \mathcal{I}$ which give the probability of choosing individual action $a^i \in \mathcal{A}^i$ after accumulating individual history $h^i \in \mathcal{H}^i$. We will use the short-hand notation $\pi_t^i(a^i|h^i)$. We can regroup these individual policies $\{\pi^i \ \forall i \in \mathcal{I}\}$ into a separable joint policy π with decision rules $\pi_t(a|h) = \prod_{i\in\mathcal{I}} \pi_t^i(a^i|h^i)$ at any step t.

Contrary to MMDPs, there is no trivial way to extract individual policies even from a deterministic joint policy because agents do not have access to the common knowledge, e.g. states or beliefs, and have their own local observations. The only sufficient statistic for individual histories known so far [33] relies on the occupancy state which is defined as follow:

Definition 3.11 (Joint occupancy states)**.** In a Dec-POMDP, the joint occupancy states are statistics summarizing the common knowledge, which are defined by:

$$o_t : s, h \mapsto \Pr\{S_t = s, H_t = h\} \qquad \forall t \in [\![0, T-1]\!] \tag{3.7}$$

Joint occupancy states can be seen as the multi-agent equivalent of the POMDP beliefs. From this joint occupancy states, we can define individual occupancy states o_t^i, which are sufficient statistics for individual histories:

Definition 3.12 (Individual occupancy states)**.** In a Dec-POMDP, the individual occupancy state for an agent $i \in \mathcal{I}$ is given by:

$$o_t^i : s, h^{-i} \mapsto \Pr\{S_t = s, H_t^{-i} = h^{-i} \mid H_t^i = h^i\} \qquad \forall t \in [\![0, T-1]\!] \tag{3.8}$$

They can be deduced from a marginalization of the joint occupancy states o_t:

$$o_t^i(s, h^{-i}, h^i) = \frac{o_t(s, h)}{\sum\limits_{\bar{s}\in\mathcal{S}} \sum\limits_{\substack{\bar{h}\in\mathcal{H}_t \\ \bar{h}^i = h^i}} o_t(\bar{s}, \bar{h})} \tag{3.9}$$

In practice, computing exact individual occupancy states is often intractable, as it requires marginalizing on all possible joint occupancy states. Models rely on summarized approximate representations, where the features composing these representations are often hand-crafted. For example, one can use truncated individual histories of the few last action-observation pairs.

Conclusion. We have now defined all the properties of decision models we will consider throughout this thesis. We can describe systems where multiple agents interact with an environment, choosing individual actions based on their own observations and actions. They cooperate towards a common objective to maximize their joint expected cumulated rewards. Something interesting to notice here is that even though the models increase in complexity, the mathematical foundations of the simple MDP can be applied to all its extensions by considering augmented state spaces. These augmented states correspond to the statistics compressing all the information available to the agents, like the beliefs in POMDP or the occupancy states in Dec-POMDPs. Hence, we will now discuss resolution methods only in the frame of MDPs. They can be extended to all other variants by adapting the representations structures and the update operators to the corresponding augmented state spaces. This adaptation is non-trivial, especially when the augmented state space is continuous, but does not question the mathematical foundations of the methods, which only rely on the Markov property.

3.2 Statistics and solution methods

Now that we have defined MDPs, MMDPs, POMDPs and Dec-POMDPs, we will review the basic solution methods for MDPs. As we discussed earlier, all models can be reframed as MDPs with augmented continuous states, and the methods described in this section can be adapted to partially observable, and/or decentralized settings. Although the mathematical foundations are pretty similar, it is non-trivial to adapt the exact algorithms we will present in this section to continuous state-space without additional structure (such as convexity of the value function). However, their heuristic variants often rely on parameterized approximation structures for the value function or the policy. They are easier to generalize to more complex problem settings, but are limited by their capacity to represent rich individual internal states on top of which we can base our decision rules. We will start by presenting some inevitable statistics when dealing with Markov Decision Processes. Then, we will review some algorithms based on them, starting with model-based ones. We will finally focus on model-free algorithms, especially ones based on the policy gradient method.

3.2.1 Bellman equations

A very important statistic in RL, and more generally in Dynamic Programming (DP), is the state value function V^π of policy π. It takes its value in $\mathbb{R}$ and evaluates the expected cumulated reward from any state $s \in \mathcal{S}$ when following π at each decision step. It is formally defined as follow:

Definition 3.13 (State Value function)**.** In a finite-horizon MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, T)$, the value of a policy π in any state s at step t is given by the state value function V^π defined as:

$$V_t^\pi : s \mapsto \mathrm{E}_{\pi,P}\left[\sum_{\tau=t}^{T-1} \gamma^{\tau-t} R(S_\tau, A_\tau, S_{\tau+1}) \mid S_t = s\right] \qquad \forall t \in [\![0, T-1]\!]$$

where $A_\tau \sim \pi_\tau(.|S_\tau)$ and $S_{\tau+1} \sim P(.|S_\tau, A_\tau)$ $\forall \tau \in [\![t, T-1]\!]$.

For infinite horizon problem ($T = \infty$), we can similarly define a stationary value function $V^\pi : \mathcal{S} \to \mathbb{R}$ which is not indexed by time. The reason why this statistic is so inevitable in RL is that it obeys the following recursive relation, known as Bellman Equation:

Theorem 3.2 (Bellman Equation [5])**.** *In a finite-horizon MDP, the state value function V^π obeys the following recursive relation:*

$$V_t^\pi(s) = \mathrm{E}_{\pi_t,P}\left[R(s, A, S') + \gamma V_{t+1}^\pi(S')\right] \qquad \forall s \in \mathcal{S} \qquad \forall t \in [\![0, T-1]\!]$$

In particular, we can focus on the optimal state value function V^*, of any optimal policy π^* maximizing our objective stated in Equation 3.1. We can verify by recursion that the following decision rules constitute an optimal deterministic policy:

$$\pi_t^* : s \mapsto \arg\max_{a \in \mathcal{A}} \mathrm{E}_P[R(s, a, S') + \gamma V_{t+1}^*(S')] \qquad \forall t \in [\![0, T-1]\!]$$

Moreover, V^* obeys the same recursive relation as any state value function, meaning we could state the following corollary to Theorem 3.2 replacing V^π by V^*, which is referred to as Bellman Optimality Equation:

Corollary 3.1 (Bellman Optimality Equation [5])**.** *In a finite-horizon MDP, the optimal state value function V^* obeys the following recursive relation:*

$$V_t^*(s) = \max_{a \in \mathcal{A}} \mathrm{E}_P \left[R(s, a, S') + \gamma V_{t+1}^*(S') \right] \qquad \forall s \in \mathcal{S} \qquad \forall t \in [\![0, T-1]\!]$$

If we add a degree of freedom on the first action a taken in state s, we obtain another statistic named the state-action value function, often shorten Q-value, because of its usual mathematical notation $Q_t^\pi(s, a)$.

Definition 3.14 (State-Action Value function)**.** In a finite-horizon MDP, the value of a policy π after taking any action a in state s at step t is given by the state-action value function Q^π defined as:

$$Q_t^\pi : s, a \mapsto \mathrm{E}_{\pi, P} \left[\sum_{\tau=t}^{T-1} \gamma^{\tau - t} R(S_\tau, A_\tau, S_{\tau+1}) \mid S_t = s, A_t = a \right] \qquad \forall t \in [\![0, T-1]\!]$$

where $A_\tau \sim \pi_\tau(\cdot|S_\tau)$ $\forall \tau \in [\![t+1, T-1]\!]$ and $S_{\tau+1} \sim P(\cdot|S_\tau, A_\tau)$ $\forall \tau \in [\![t, T-1]\!]$.

The Q-value function also obeys a Bellman equation similarly to Theorem 3.2:

Theorem 3.3 (Bellman equation for Q-Value)**.** *The Q-Value function Q^π obeys the following recursive relation:*

$$Q_t^\pi(s, a) = \mathrm{E}_{P, \pi}[R(s, a, S') + \gamma Q_{t+1}^\pi(S', A')] \quad \forall s \in \mathcal{S} \quad \forall a \in \mathcal{A} \quad \forall t \in [\![0, T-1]\!] \tag{3.10}$$

and, similarly to the optimal value function, the optimal Q-value function Q^* obeys:

$$Q_t^*(s, a) = \mathrm{E}_P[R(s, a, S') + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}^*(S', a')] \quad \forall s \in \mathcal{S} \quad \forall a \in \mathcal{A} \quad \forall t \in [\![0, T-1]\!] \tag{3.11}$$

3.2.2 Value-based algorithms

Corollary 3.1 leads to the following iterative algorithm (Algorithm 3.1) called `Value Iteration`, that converge towards $V \to V^*$, from which we can extract an optimal deterministic policy π^*:

foreach $s \in \mathcal{S}, t \in [\![0, T]\!]$ **do** $V_t(s) \leftarrow 0$
while *stopping criterion not met* **do**
  foreach $s \in \mathcal{S}, t \in [\![0, T-1]\!]$ **do**
    $V_t(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \big(R(s, a, s') + \gamma\, V_{t+1}(s') \big)$
foreach $s \in \mathcal{S}, t \in [\![0, T-1]\!]$ **do**
  $\pi_t^*(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \big(R(s, a, s') + \gamma V_{t+1}(s') \big)$

Algorithm 3.1: Value Iteration

In the Operation Research literature, the `Value Iteration` algorithm is often simply called `Dynamic Programming`, and its approximation have been successfully used to address some variants of VRP [83, 100, 16].

If it is capable of converging towards optimal policies, `Value Iteration` requires to know the transition function P and the reward function R in order to compute the expected value over the possible next states while applying the Bellman update operator and the policy extraction. From now on, we will review what are referred to as model-free algorithms. They do not require to know the model a priori, but rely on a way to collect trajectories of interactions between the agent and its environment.

foreach $s \in \mathcal{S}, a \in \mathcal{A}, t \in [\![0, T]\!]$ **do** $Q_t(s, a) \leftarrow 0$
while *stopping criterion not met* **do**
  trajectory $\leftarrow$ `sample_trajectory(`Q`)`
  foreach t, $(s, a, r, s') \in$ `enumerate(trajectory)` **do**
    $\delta_t(s, a) \leftarrow \big(r + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}(s', a')\big) - Q_t(s, a)$
    $Q_t(s, a) \leftarrow Q_t(s, a) + \alpha.\delta_t(s, a)$

Algorithm 3.2: Q-Learning

The `SARSA` algorithm is almost identical to `Q-Learning` as described in Algorithm 3.2, except that when computing the TD error δ, the action a' maximizing the Q-value in the next state is replaced by the next action a' taken by the agent in the trajectory (hence the name: **foreach** $(s, a, r, s', a') \in$ trajectory **do** ...). Given a proper exploration policy and enough training iterations, `Q-Learning` and `SARSA` are both guaranteed to converge towards an optimal policy.

3.2.3 Policy-based algorithms

This next algorithm follows an approach quite different of the one of `Q-Learning`. We now consider a non-stationary stochastic policy composed of decision rules $\pi_t(a|s; \boldsymbol{\theta}_t)$ parameterized by weights $\boldsymbol{\theta}_t \in \mathbb{R}^{d_\pi}$ $\forall t \in [\![0, T-1]\!]$, where $d_\pi \in \mathbb{N}$ is the dimension of the parameter vector θ, i.e. the number of parameters that drive the policy. We regroup parameters of π at all time steps in the vector $\boldsymbol{\theta} \in \mathbb{R}^{T.d_\pi}$ by concatenation. The idea is to use gradient descent on the objective to directly update the parameters of the policy. This gradient is given by the following theorem:

Theorem 3.4 (Policy Gradient Theorem [106])**.** *For any finite horizon MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, T)$, and any non-stationary parameterized policy $\pi(\cdot; \boldsymbol{\theta})$, the partial derivative of the objective value w.r.t. the policy parameters is given by the following relation:*

$$\frac{\partial \, \mathrm{E}_{P_0}\left[V_0^\pi(S_0)\right]}{\partial \boldsymbol{\theta}} = \mathrm{E}_{\pi,P,P_0}\left[\sum_{t=0}^{T-1} \frac{\partial \log \pi_t(A_t|S_t; \boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \sum_{t=0}^{T-1} \gamma^t R(S_t, A_t, S_{t+1})\right] \tag{3.12}$$

$$= \mathrm{E}_{\pi,P,P_0}\left[\sum_{t=0}^{T-1} \frac{\partial \log \pi_t(A_t|S_t; \boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \gamma^t Q_t^\pi(S_t, A_t)\right] \tag{3.13}$$

Equation (3.12) was previously used by Williams [117] to develop the `REINFORCE` algorithm, which simply consists in a (stochastic) gradient descent using trajectory samples to approximate the gradient of the objective function w.r.t. the parameters $\boldsymbol{\theta}$ of the policy. It converges towards a local optimum and suffers from the high variance of the reward accumulated during the whole trajectory.

To reduce the variance of the gradient approximation, we can replace the exact policy gradient stated above in Theorem 3.4 by an unbiased estimation using an approximate Q-value function $\widehat{Q}_t^\pi(s, a; \boldsymbol{w}_t)$, parameterized by weights regrouped by concatenation in the vector $\boldsymbol{w} \in \mathbb{R}^{T.d_Q}$,

where $d_Q \in \mathbb{N}$ is the number of parameters characterizing the Q-Value approximation for each step. To guarantee that the gradient estimation stays unbiased, the following sufficient condition should be enforced:

Theorem 3.5 (Critic compatibility [106])**.**

$$\frac{\partial \widehat{Q}_t^{\pi}(s,a;\boldsymbol{w}_t)}{\partial \boldsymbol{w}} = \frac{\partial \log \pi_t(a|s;\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \qquad \forall t \in [\![0, T-1]\!], s \in \mathcal{S}, a \in \mathcal{A} \tag{3.14}$$

Sutton et al. suggest that it might in fact also be a necessary condition.

It yields the following `Actor-Critic` algorithm (Algorithm 3.3) where an "actor" trains the policy based on the guidance of an approximate Q-value representation, which is itself trained at the same time by a "critic". To lighten the notation, we will write $\boldsymbol{\Phi}_t(s,a;\boldsymbol{\theta}) = \frac{\partial \log \pi_t(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{d_\pi}$. The hyper-parameters $\alpha \in [0,1]$ and $\beta \in [0,1]$ are the learning rates of the actor and the critic, respectively.

```
θ, w ← initialize_parameters()
while stopping criterion not met do
    Δθ, Δw ← 0
    trajectory ← sample_trajectory(θ)
    foreach t, (s, a, r, s') ∈ enumerate(trajectory) do
        δ ← (r + γ max_a' q_value(s',a',t+1,w)) − q_value(s,a,t,w)
        Δw ← Δw + Φ_t(s, a, θ).δ
        Δθ ← Δθ + Φ_t(s, a, θ).γ^t.q_value(s,a,t,w)
    θ ← θ + α.Δθ
    w ← w + β.Δw
```

Algorithm 3.3: Actor-Critic

3.3 Centralized Training for Decentralized Control

In many real-world cooperative multi-agent systems, conditions at the training phase do not need to be as strict as those at the execution phase. During rehearsal, for example, actors can read the script, take breaks, or receive feedback from the director, but none of these will be possible during the show. To win matches, a soccer coach develops tactics before the games, that players will apply during the game. This asymmetry of information available during training and at execution has not been highly investigated for model-free Reinforcement Learning method, and it is natural to wonder whether the policy gradient approach in such a paradigm could be as successful as for the single-agent case.

We call this paradigm Centralized Training for Decentralized Control (CTDC). It has already been quite successful in planning [44, 109, 2, 75, 32]. More recently, (model-free) learning-based methods [33, 60, 51] have also demonstrated that CTDC offers somme advantages over more traditional, fully decentralized, approaches. In this CTDC paradigm, we have access to more information during training and we can use the common knowledge (states or beliefs) that agents normally cannot observe. Nonetheless, CTDC still preserves the ability of all agents to act independently at the execution phase.

While most previous works on multi-agent policy gradient method, such as [80], belongs to the Decentralized Training for Decentralized Control (DTDC) paradigm, also called Independent Learners, [37, 42, 61] demonstrated successful application of this new CTDC paradigm with policy gradient methods applied to Deep Neural Networks. Figure 3.4 illustrates the difference between an implementation of actor-critic in each of the two paradigms. Despite these promising existing works, the mathematical foundations of multi-agent RL are still in their infancy. We start with some of our theoretical results that contribute to extend these foundations. From it, we deduce an algorithm called Actor Critic for Decentralized Control (ACDC) that exploits the properties we highlighted. We then conclude on some experiments using two different internal state representation using some standard benchmarks of Dec-POMDPs. This section describes one of the contribution of this thesis, as a preliminary work on multi-agent sequential decision making. It was published in [2] and [4], with complementary information available in the companion research report [6].

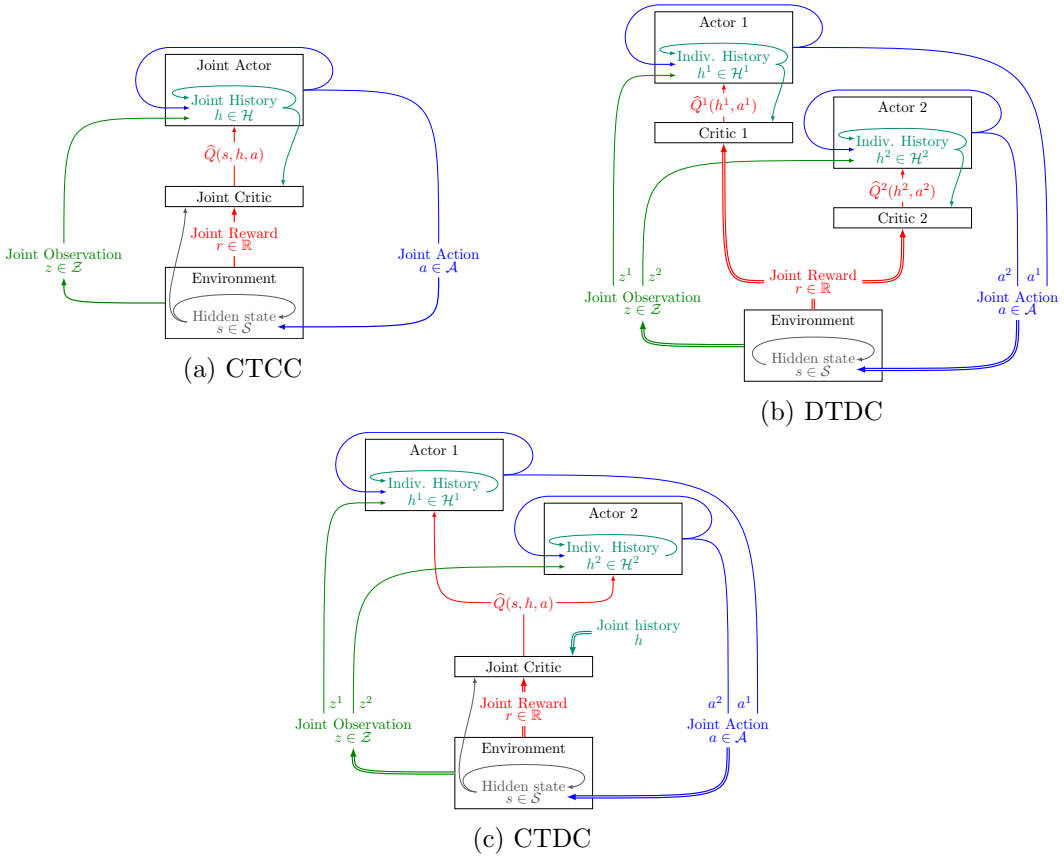


Figure 3.4 – Comparison between actor-critic in the fully centralized paradigm (a) on the left, the fully decentralized paradigm (b) on the right (also called independent learners), and the CTDC paradigm (c) in the middle. While the independent learners have to build individual statistics to drive their optimization, the extra information we have available during training in CTDC makes it possible to maintain a joint statistic that help avoid local minima and can lead the agents to coordinate themselves better when executing their individual policies in a decentralized way.

3.3.1 Multi-Agent Policy Gradient Theorem

Our primary result is an extension of the policy gradient theorem [106] presented in Theorem 3.4 for MDPs to Dec-POMDPs. For now, we will consider non-stationary parametric individual policies π^i for every agent $i \in \mathcal{I}$. Every individual decision rule at time $t \in [\![0, T-1]\!]$ is parameterized by a vector $\boldsymbol{\theta}_t^i \in \mathbb{R}^{d_\pi}$. As a notation shortcut, we can concatenate all these vectors of parameters to get a single vector $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{I}|.T.d_\pi}$ for the joint policy π.

The joint state-action value function Q^π of our Dec-POMDP can be rewritten from Definition 3.14 as a function of the state, the joint history and the joint action:

$$Q_t^\pi : s, h, a \mapsto \mathrm{E}_{\pi,P,\Omega}\left[\sum_{\tau=t}^{T-1} \gamma^{\tau-t} R(S_t, A_t, S_t') \mid S_t = s, H_t = h, A_t = a\right] \qquad \forall t \in [\![0, T-1]\!]$$

Similarly, we can redefine a joint state value function V, which is also augmented with the joint history:

$$V_t^\pi : s, h \mapsto \mathrm{E}_{\pi,P,\Omega}\left[\sum_{\tau=t}^{T-1} \gamma^{\tau-t} R(S_t, A_t, S_t') \mid S_t = s, H_t = h\right] \qquad \forall t \in [\![0, T-1]\!]$$

The Policy Gradient Theorem adapted to the multi-agent case can now be stated as follow:

Theorem 3.6 (Multi-agent Policy Gradient)**.** *For any finite horizon Dec-POMDP $(\mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{Z}, P, \Omega, R, P_0, T)$, and any separable non-stationary parameterized policy $\pi(\cdot;\boldsymbol{\theta})$, the partial derivative of the objective value w.r.t. the policy parameters is given by the following relation:*

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\Omega,P_0}\left[V_0^\pi(S_0, H_0)\right] = \mathrm{E}_{\pi,P,\Omega,P_0}\left[\sum_{t=0}^{T-1}\sum_{i\in\mathcal{I}} \frac{\partial \log \pi_t^i(A_t^i|H_t^i;\boldsymbol{\theta}_t^i)}{\partial \boldsymbol{\theta}} \gamma^t Q_t^\pi(S_t, H_t, A_t)\right]$$

Proof of Theorem 3.6. Starting from the Bellman equation for Q_t^π:

$$Q_t^\pi(s,h,a) = \mathrm{E}_P\left[R(s,a,S')\right] + \gamma\, \mathrm{E}_{\pi_{t+1},P,\Omega}\left[Q_{t+1}^\pi(S', H', A')\right]$$

we take partial derivative and write the expected value w.r.t. π_{t+1} explicitly:

$$\Leftrightarrow \frac{\partial Q_t^\pi(s,h,a)}{\partial \boldsymbol{\theta}} = \gamma\, \mathrm{E}_{P,\Omega}\left[\sum_{a'\in\mathcal{A}} \frac{\partial}{\partial \boldsymbol{\theta}}\left(\pi_{t+1}(a'|H') Q_{t+1}^\pi(S', H', a')\right)\right]$$

expend the partial derivative:

$$= \gamma\, \mathrm{E}_{P,\Omega}\left[\sum_{a'\in\mathcal{A}} \frac{\partial \pi_{t+1}(a'|H')}{\partial \boldsymbol{\theta}} Q_{t+1}^\pi(S', H', a') + \pi_{t+1}(a'|H') \frac{\partial Q_{t+1}^\pi(S', H', a')}{\partial \boldsymbol{\theta}}\right]$$

use a log trick to factor π_{t+1}:

$$\Leftrightarrow \frac{\partial Q_t^\pi(s,h,a)}{\partial \boldsymbol{\theta}} = \gamma\, \mathrm{E}_{\pi_{t+1}P,\Omega}\left[\frac{\partial \log \pi_{t+1}(A'|H')}{\partial \boldsymbol{\theta}} Q_{t+1}^\pi(S', H', A') + \frac{\partial Q_{t+1}^\pi(S', H', A')}{\partial \boldsymbol{\theta}}\right]$$

split π_{t+1} into independent individual decision rules:

$$= \gamma \, \mathrm{E}_{\pi_{t+1} P, \Omega} \left[\sum_{i \in \mathcal{I}} \frac{\partial \log \pi^i_{t+1}(A'^i | H'^i)}{\partial \boldsymbol{\theta}} Q^\pi_{t+1}(S', H', A') + \frac{\partial Q^\pi_{t+1}(S', H', A')}{\partial \boldsymbol{\theta}} \right]$$

We can now apply this recursive relation to our objective until we obtain the final result:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\Omega, P_0} \left[V^\pi_0(S_0, H_0) \right] &= \frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\pi_0, \Omega, P_0} \left[Q^\pi_0(S_0, H_0, A_0) \right] \\
&= \mathrm{E}_{\pi_0, \Omega, P_0} \left[\sum_{i \in \mathcal{I}} \frac{\partial \log \pi^i_0(A^i_0 | H^i_0)}{\partial \boldsymbol{\theta}} Q^\pi_0(S_0, H_0, A_0) + \frac{\partial Q^\pi_0(S_0, H_0, A_0)}{\partial \boldsymbol{\theta}} \right] \\
&= \mathrm{E}_{\pi_0, \pi_1, P, \Omega, P_0} \left[\sum_{i \in \mathcal{I}} \frac{\partial \log \pi^i_0(A^i_0 | H^i_0)}{\partial \boldsymbol{\theta}} Q^\pi_0(S_0, H_0, A_0) \right. \\
&\qquad \left. + \gamma \frac{\partial \log \pi^i_1(A^i_1 | H^i_1)}{\partial \boldsymbol{\theta}} Q^\pi_1(S_1, H_1, A_1) + \gamma \frac{\partial Q^\pi_1(S_1, H_1, A_1)}{\partial \boldsymbol{\theta}} \right] \\
&= \dots \\
&= \mathrm{E}_{\pi, P, \Omega, P_0} \left[\sum_{t=0}^{T-1} \sum_{i \in \mathcal{I}} \frac{\partial \log \pi^i_t(A^i_t | H^i_t)}{\partial \boldsymbol{\theta}} \gamma^t Q^\pi_t(S_t, H_t, A_t) \right]
\end{aligned}$$

□

Additionally, the same relation holds if we replace the Q-value of the policy by its advantage A^π, which is defined as:

$$A^\pi_t(s, h, a) = Q^\pi_t(s, h, a) - V^\pi_t(s, h) \qquad \forall t \in [\![0, T-1]\!], s \in \mathcal{S}, h \in \mathcal{H}_t, a \in \mathcal{A}$$

and gives the relative benefit of choosing a specific action a instead of following policy π in state s after history h at time t. We get the following corollary to Theorem 3.6:

Corollary 3.2 (Multi-agent policy gradient with advantage)**.** *With the same hypotheses as Theorem 3.6, the partial derivative of the objective value w.r.t. the policy parameters is also given by the following relation:*

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\Omega, P_0} \left[V^\pi_0(S_0, H_0) \right] = \mathrm{E}_{\pi, P, \Omega, P_0} \left[\sum_{t=0}^{T-1} \sum_{i \in \mathcal{I}} \frac{\partial \log \pi^i_t(A^i_t | H^i_t; \boldsymbol{\theta}^i_t)}{\partial \boldsymbol{\theta}} \gamma^t A^\pi_t(S_t, H_t, A_t) \right]$$

Proof of Corollary 3.2. The proof starts by reverting the split into individual policies and the log trick in the following expression, which holds $\forall t \in [\![0, T-1]\!], s \in \mathcal{S}, h \in \mathcal{H}_t$:

$$\begin{aligned}
\mathrm{E}_{\pi_t} \left[\sum_{i \in \mathcal{I}} \frac{\partial \log \pi^i_t(A^i_t | h^i_t)}{\partial \boldsymbol{\theta}} \gamma^t V^\pi_t(s, h) \right] &= \mathrm{E}_{\pi_t} \left[\frac{\partial \log \pi_t(A_t | h_t)}{\partial \boldsymbol{\theta}} \gamma^t V^\pi_t(s, h) \right] \\
&= \sum_{a \in \mathcal{A}} \frac{\partial \pi_t(a|h)}{\partial \boldsymbol{\theta}} \gamma^t V^\pi_t(s, h) \\
&= \frac{\partial}{\partial \boldsymbol{\theta}} \left(\sum_{a \in \mathcal{A}} \pi_t(a|h) \right) \gamma^t V^\pi_t(s, h) \\
&= \mathbf{0}
\end{aligned}$$

We can subtract all these null terms to the expression of Theorem 3.6 for each time step to get the result. □

3.3.2 Critic compatibility

Now, similarly to the single agent case, we will discuss the problem of critic compatibility when replacing the exact Q-value of the policy by an approximation $\widehat{Q}$ parameterized by some vectors $\boldsymbol{w}_t \in \mathbb{R}^{d_Q}\ \forall t \in [\![0, T-1]\!]$, which we can regroup into a single vector $\boldsymbol{w} \in \mathbb{R}^{T.d_Q}$ by concatenation. To guarantee that the gradient approximation is unbiased, we have the following sufficient condition:

Theorem 3.7 (Multi-agent compatible critic)**.** *If the parameters $\boldsymbol{w}$ minimize the mean square error between Q^π and $\widehat{Q}$, and if, $\forall t \in [\![0, T-1]\!], s \in \mathcal{S}, h \in \mathcal{H}_t, a \in \mathcal{A}$, the following condition holds:*

$$\widehat{Q}_t(s,h,a) = \widehat{A}_t(s,h,a;\boldsymbol{w}_t^A) + \widehat{V}_t(s,h;\boldsymbol{w}_t^V) \qquad \textit{where} \qquad \boldsymbol{w}_t^A \amalg \boldsymbol{w}_t^V = \boldsymbol{w}_t$$
$$s.t.\ \frac{\partial \widehat{A}_t(s,h,a;\boldsymbol{w}_t^A)}{\partial \boldsymbol{w}_t^A} = \sum_{i\in\mathcal{I}} \frac{\partial \log \pi_t^i(a^i|h^i;\boldsymbol{\theta}_t^i)}{\partial \boldsymbol{\theta}_t}$$

then the gradient of the objective w.r.t. the policy parameters as stated in Theorem 3.6 is preserved when replacing Q^π by its parameterized approximation $\widehat{Q}$.

Proof of Theorem 3.7. If $\boldsymbol{w}$ minimize the mean square error between Q^π and $\widehat{Q}$, it means that $\forall t \in [\![0, T-1]\!]$:

$$\frac{\partial}{\partial \boldsymbol{w}} \mathrm{E}_{\pi,P,\Omega,P_0}\left[\left(Q_t^\pi(S_t,H_t,A_t) - \widehat{Q}_t(S_t,H_t,A_t)\right)^2\right] = \mathbf{0}$$
$$\Rightarrow \mathrm{E}_{\pi,P,\Omega,P_0}\left[\frac{\partial \widehat{Q}_t^\pi(S_t,H_t,A_t)}{\partial \boldsymbol{w}}\left(Q_t^\pi(S_t,H_t,A_t) - \widehat{Q}_t(S_t,H_t,A_t)\right)\right] = \mathbf{0}$$

focusing on some components of $\boldsymbol{w}$ this means in particular that:

$$\Rightarrow \mathrm{E}_{\pi,P,\Omega,P_0}\left[\frac{\partial \widehat{Q}_t(S_t,H_t,A_t)}{\partial \boldsymbol{w}_t^A}\left(Q_t^\pi(S_t,H_t,A_t) - \widehat{Q}_t(S_t,H_t,A_t)\right)\right] = \mathbf{0}$$
$$\Rightarrow \mathrm{E}_{\pi,P,\Omega,P_0}\left[\frac{\partial \widehat{A}_t(S_t,H_t,A_t)}{\partial \boldsymbol{w}_t^A} Q_t^\pi(S_t,H_t,A_t)\right] = \mathrm{E}_{\pi,P,\Omega,P_0}\left[\frac{\partial \widehat{A}_t(S_t,H_t,A_t)}{\partial \boldsymbol{w}_t^A} \widehat{Q}_t(S_t,H_t,A_t)\right]$$

if we introduce the condition stated in Theorem 3.7, we get:

$$\mathrm{E}_{\pi,P,\Omega,P_0}\left[\sum_{i\in\mathcal{I}} \frac{\partial \log \pi_t^i(a^i|h^i)}{\partial \boldsymbol{\theta}_t} Q_t^\pi(S_t,H_t,A_t)\right] = \mathrm{E}_{\pi,P,\Omega,P_0}\left[\sum_{i\in\mathcal{I}} \frac{\partial \log \pi_t^i(a^i|h^i)}{\partial \boldsymbol{\theta}_t} \widehat{Q}_t^\pi(S_t,H_t,A_t)\right]$$

We can now sum on t to recover the expression of Theorem 3.7 on the left, and obtain the final result:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathrm{E}_{\Omega,P_0}\left[V_0^\pi(S_0,H_0)\right] = \mathrm{E}_{\pi,P,\Omega,P_0}\left[\sum_{t=0}^{T-1}\sum_{i\in\mathcal{I}} \frac{\partial \log \pi_t^i(a^i|h^i)}{\partial \boldsymbol{\theta}} \widehat{Q}_t^\pi(S_t,H_t,A_t)\right]$$

□

It is interesting to note that the vector of partial derivatives $\frac{\partial \log \pi_t^i(a^i|h^i;\boldsymbol{\theta}_t^i)}{\partial \boldsymbol{\theta}}$ is mostly zeros except for the block corresponding to $\frac{\partial \log \pi_t^i(a^i|h^i;\boldsymbol{\theta}_t^i)}{\partial \boldsymbol{\theta}_t^i}$. This structural property combined with the sufficient condition in Theorem 3.7 has a very interesting consequence on the form of the compatible critic $\widehat{Q}$: its advantage term is separable into individual components, while its value term offers some degree of freedom to incorporate additional information available only during the centralized training phase:

$$\widehat{Q}_t(s,h,a;\boldsymbol{w}_t) = \sum_{i\in\mathcal{I}} \widehat{A}_t^i(h^i,a^i;\boldsymbol{w}_t^i) + \widehat{V}_t(s,h;\boldsymbol{w}_t^V) \qquad \forall t \in [\![0,T-1]\!], s\in\mathcal{S}, h\in\mathcal{H}_t, a\in\mathcal{A}$$

$$\text{where } \widehat{A}_t^i(h^i,a^i) = \left(\frac{\partial \log \pi_t^i(a^i|h^i)}{\partial \boldsymbol{\theta}_t^i}\right)^\intercal \boldsymbol{w}_t^i \qquad \forall i \in \mathcal{I}$$

3.3.3 Experimental validation

To evaluate the benefit of considering the CTDC paradigm with Policy Gradient methods, we have implemented a variant of Actor-Critic algorithm based on the theoretical results we stated in Theorems 3.6 and 3.7. We call this algorithm *Actor-Critic for Decentralized Control*, or `ACDC` for short, for which we provide pseudo-code in Algorithm 3.4. The major difference with more classical Actor-Critic lies in the structure of both the actor (which is decentralized, i.e. with separable individual policies) and the critic (which meets the sufficient condition for compatibility stated above). We also use mini-batches of trajectories to reduce the variance in each parameters update. Similarly to what we have done before to lighten the notation, we write:

$$\boldsymbol{\Phi}_t^i(h^i,a^i) = \frac{\partial \log \pi_t^i(a^i|h^i;\boldsymbol{\theta}_t^i)}{\partial \boldsymbol{\theta}_t^i} = \frac{\partial \widehat{A}_t^i(h^i,a^i;\boldsymbol{w}_t^i)}{\partial \boldsymbol{w}_t^i} \in \mathbb{R}^{d_\pi}$$

$$\text{and } \boldsymbol{\Phi}_t^V(s,h) = \frac{\partial \widehat{V}_t(s,h;\boldsymbol{w}_t^V)}{\partial \boldsymbol{w}_t^V} \in \mathbb{R}^{d_V}$$

There are many key components in actor-critic methods that can affect their performances. These key components include:

- training paradigms (CTCC *vs* DTDC *vs* CTDC);
- policy representations (stationary *vs* non-stationary policies);
- approximation architectures (linear *vs* Recurrent Neural Networks (RNN));
- history representations (truncated histories *vs* hidden states of RNN).

We implemented three variants of actor-critic methods that combine these components. Unless otherwise mentioned, we will refer to actor-critic methods from: the acronym of the paradigm in which they have been implemented, *e.g.,* CTDC for ACDC; plus the key components, "*CTDC_TRUNC(K)*" for `ACDC` where we use K last observations instead of histories as inputs of a tabular non-stationary policy; or "*DTDC_RNN*" for distributed `Reinforce` where we use RNNs as a stationary policy, see Figure 3.5.

$\boldsymbol{\theta}, \boldsymbol{w} \leftarrow$ `initialize_parameters()`
while *stopping criterion not met* **do**
  $\Delta\boldsymbol{\theta}, \Delta\boldsymbol{w} \leftarrow \mathbf{0}$
  foreach *mini-batch element* **do**
    trajectory $\leftarrow$ `sample_trajectory(`$\boldsymbol{\theta}$`)`
    $s \leftarrow s_0$
    foreach $i \in \mathcal{I}$ **do** $h^i \leftarrow$ `initialize_history_repr(`z_0^i`)`
    foreach t, $(a, r, s', z') \in$ `enumerate(trajectory)` **do**
      foreach $i \in \mathcal{I}$ **do** $h'^i \leftarrow$ `update_history_repr(`h^i, a^i, z'^i`)`
      $\delta \leftarrow \left(r + \gamma \max_{a'} \widehat{Q}_{t+1}(s', h', a'; \boldsymbol{w}_{t+1})\right) - \widehat{Q}_t(s, h, a; \boldsymbol{w}_t)$
      $\Delta\boldsymbol{w}_t^V \leftarrow \Delta\boldsymbol{w}_t^V + \boldsymbol{\Phi}_t^V(s, h).\delta$
      foreach $i \in \mathcal{I}$ **do**
        $\Delta\boldsymbol{w}_t^i \leftarrow \Delta\boldsymbol{w}_t^i + \boldsymbol{\Phi}_t^i(h^i, a^i).\delta$
        $\Delta\boldsymbol{\theta}_t^i \leftarrow \Delta\boldsymbol{\theta}_t^i + \boldsymbol{\Phi}_t^i(h^i, a^i)\gamma^t \widehat{Q}_t(s, h, a; \boldsymbol{w}_t)$
      $s \leftarrow s'$
      foreach $i \in \mathcal{I}$ **do** $h^i \leftarrow h'^i$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha.\Delta\boldsymbol{\theta}$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta.\Delta\boldsymbol{w}$

Algorithm 3.4: Actor-Critic for Decentralized Control (ACDC)

We run simulations on standard benchmarks from Dec-POMDP literature, including *Dec. Tiger*, *Broadcast Channel*, *Mars*, *Box Pushing*, *Meeting in a Grid*, and *Recycling Robots*, which can be found on http://masplan.org.

Broadcast Channel Two agents want to communicate through a shared communication channel. They have to learn a strategy, i.e. a communication protocol, to avoid sending messages all at once. There are 4 states, 2 actions, and 2 observations.

Recycling Robots Two robots have to collect small and large cans while watching out their battery level. They can pickup small cans alone but must cooperate for large can. Actions are abstracted to the level of "searching for cans" and "recharge".

Meeting in a Grid Two robots evolve in a small 3×3 grid world and must meet on the top left or bottom right corner. They can observe the walls around them, and move in all four directions or stay in place.

Box Pushing Two robots have to move two small and one large boxes to some target positions on a grid. They can only see what is on the cell right in front of them, then move forward, turn on the spot, or stay in place. They can push the small boxes alone but must cooperate to move the large one.

Mars Two rovers must conduct a set of experiments on different sites on Mars. It can be seen as an advanced version of the recycling robots problem, with more states, actions and observations.

Dec. Tiger Two agents needs to open one of two doors avoiding a hidden tiger. They can listen for roars, but cannot communicate with one another. They need to listen multiple time to confirm where they hear the tiger from, and need some form of memory in their internal representation.

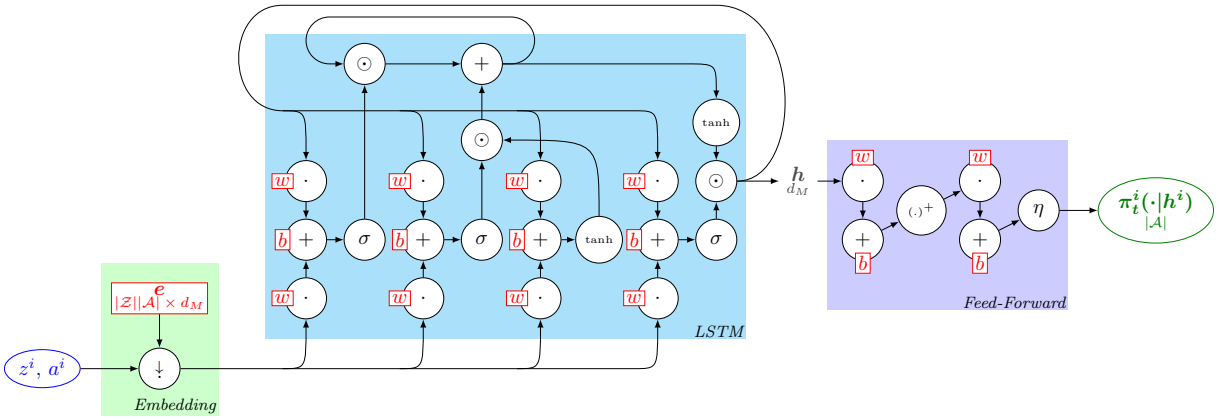


Figure 3.5 – Individual policy architecture using LSTM to compress history and maintain an individual internal state for each agent.

History Representation Matters

In this section, we conducted experiments with the goal of gaining insights on how the representation of histories affects the performance of ACDC methods. Figure 3.6 depicts the comparison of truncated histories *vs* hidden states of deep neural networks. Results obtained using an ϵ-optimal planning algorithm called `FB-HSVI` [33] are included as reference. For short planning horizons, *e.g.*, $T = 10$, *CTDC_RNN* quickly converges to good solutions in comparison to *CTDC_TRUNC(1)* and *CTDC_TRUNC(3)*. This suggests *CTDC_RNN* learns more useful and concise representations of histories than the truncated representation. However, for some of the more complex tasks such as *Dec. Tiger*, *Box Pushing* or *Mars*, no internal representation was able to perform optimally.

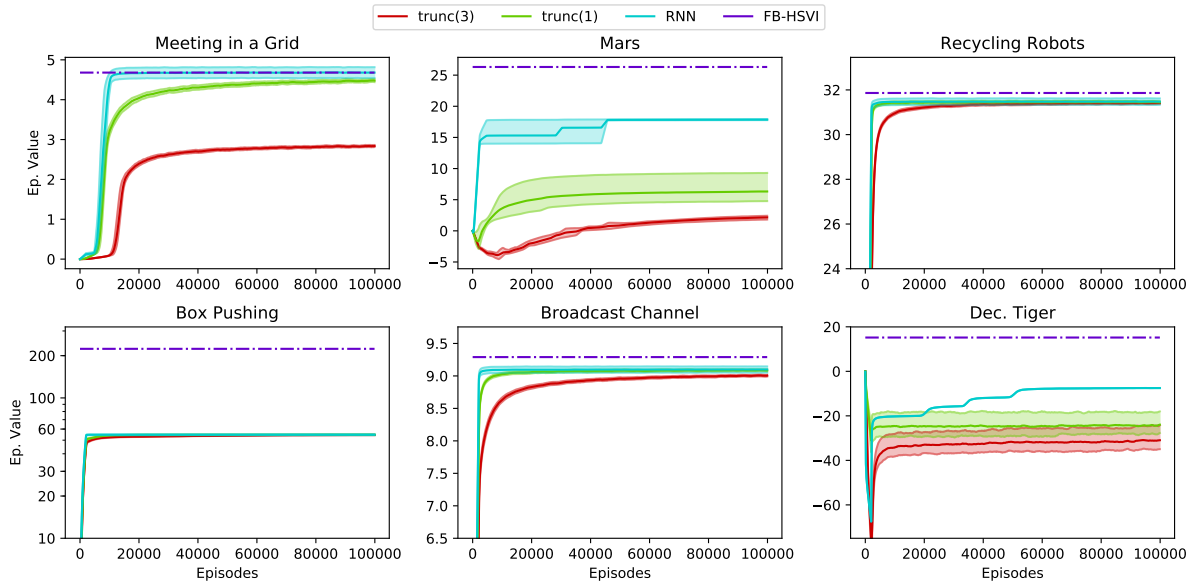


Figure 3.6 – Comparison of different structures used to represent histories.

Overall, our experiments on history representations show promising results for RNNs, which have the advantage over truncated histories to automatically learn equivalence classes and compact internal representations based on the gradient back-propagated from the reward signal.

Care should be taken though, as some domain planning horizons and other specific properties might cause early convergence to poor local optima. We are not entirely sure which specific features of the problems deteriorate performances, and we leave for future works to explore better methods to train these architectures.

Comparing Paradigms Altogether

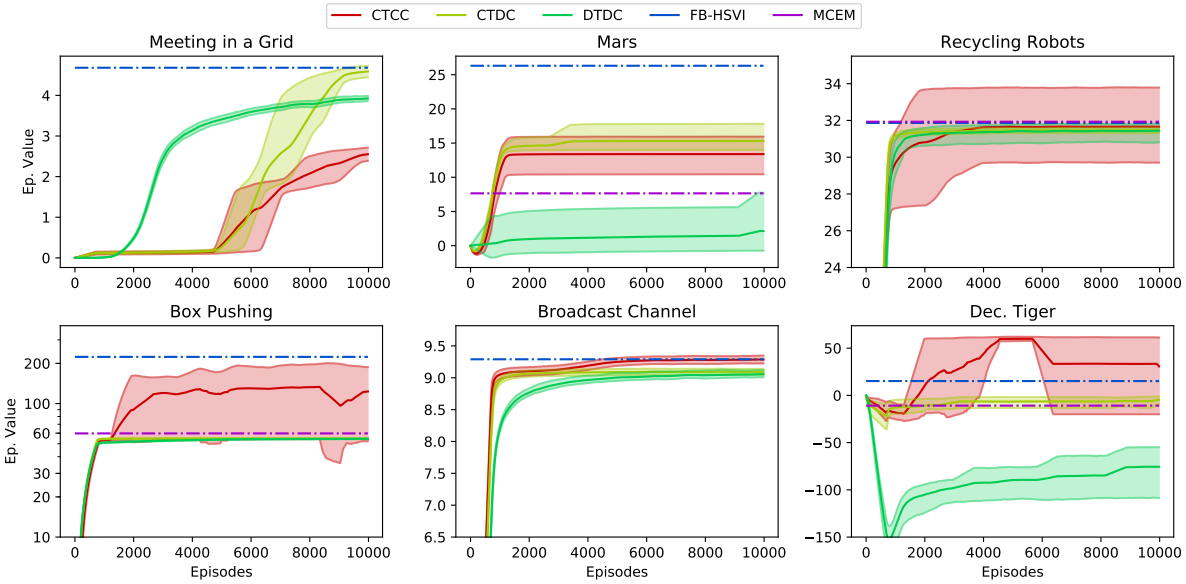


Figure 3.7 – Comparison of the three paradigms for $T = 10$.

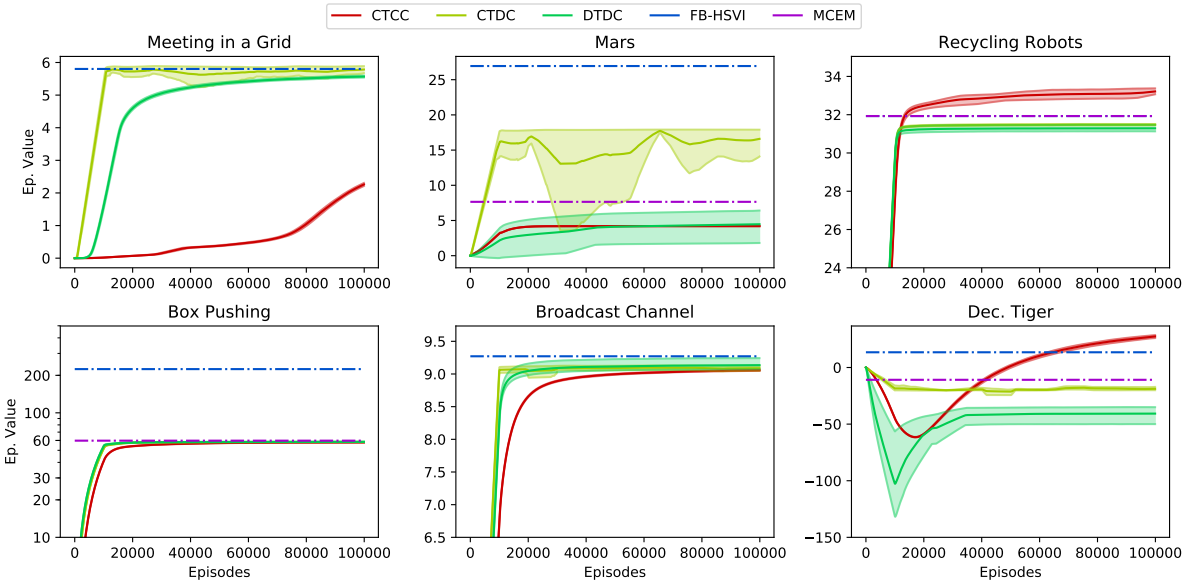


Figure 3.8 – Comparison of the three paradigms for $T = \infty$.

In this experiments, we compare paradigms, CTCC, DTDC, and CTDC using our RNN approximation structure. We complement our experiments with results from other Dec-POMDP algorithms: an ϵ-optimal planning algorithm called `FB-HSVI` [33]; and a sampling-based planning

algorithm called Monte-Carlo Expectation-Maximization (`MCEM`) algorithm [118], which shares many similarities with actor-critic methods. It is worth noticing that we are not competing against FB-HSVI as it is model-based. As for MCEM, we reported performances[1] recorded in [118].

In almost all tested benchmarks, CTDC seems to take the better out of the two other paradigms, for either $T = 10$ (Fig. 3.7) or $T = \infty$ (Fig. 3.8). CTCC might suffer from the high dimensionality of the joint history space, and fail to explore it efficiently before the learning step-sizes become negligible, or we reached the predefined number of training episodes. Our on-policy sampling evaluation certainly amplified this effect. Having a much smaller history space to explore, CTDC outperforms CTCC in these experiments. Compared to DTDC which also explores smaller history space, there is a net gain to consider a compatible centralized critic in the CTDC paradigm, resulting in better performances. Even if CTDC achieves performances better or equal to the state of the art MCEM algorithm, there is still some margins of improvements to reach the global optima given by FB-HSVI in every benchmark. As previously mentioned, this is partly due to inefficient representations of histories.

3.4 Conclusion

In this chapter, we introduced the basics of sequential decision making modelled as Markov Decision Processes, which are naturally designed to describe stochastic and dynamic interactions between an agent and its environment, hence our interest when considering online stochastic vehicle routing problems. We have reviewed some standard approaches to find optimal policies which define how the agent should act to accumulate some reward signal over time. We have then extended this model to the partially observable, and the multi-agent decentralized settings. If we mentioned some statistics that are capable of compressing the information available to every agent involved, we also insisted on the core problem of internal state representation in Reinforcement Learning, especially when using model-free approaches.

We then described some preliminary theoretical contributions that focused on a somewhat new paradigm where training is conducted under favorable circumstances that provide more information than what will be available to each agent at execution time, while still preserving the capability to act independently later on. We extended known results of Actor-Critic methods from MDPs to Dec-POMDPs in this Centralized Training for Decentralized Control paradigm, and exposed some interesting structural properties of the different functions involved. We evaluated our approach against other standard paradigms, and reported competitive and promising results. We also compared different individual history representation and compression functions, which demonstrated that Deep Neural Networks, and more particularly recurrent ones, can learn quite efficient individual internal state representation.

[1] Two results in MCEM [118] were above optimal values, so we reported optimal values instead.

Chapter 4

Sequential Multi-agent Markov Decision Process

In Chapter 2 we presented the state of the art in Vehicle Routing Problems (VRPs), especially the Dynamic and Stochastic variants (DS-VRPs), where the information available to make decisions evolves along the route during execution and is subject to uncertainty. We saw that classical formalization based on Mixed Integer Linear Program (MILP) or Constraint Programming (CP) are not well adapted to describe such dynamic and stochastic problems.

That is why we presented in Chapter 3 some background knowledge on Markov Decision Processes (MDPs) and Reinforcement Learning (RL), which naturally describe dynamic decision-making processes under uncertainty. We focussed on multi-agent models, such as the more general Decentralized POMDPs (Dec-POMDPs) or the simpler Multi-agent MDPs (MMDPs). Indeed, the systems we are interested in when dealing with DS-VRPs involve many vehicles which we would like to consider as autonomous agents. However, we wanted to take into account that more information is available while training our agents than when they will execute their learned policies. To study this information asymmetry, we introduced a learning paradigm we call Centralized Training for Decentralized Control (CTDC) and contributed to the development of Policy Gradient methods in this paradigm.

In this chapter, we will develop another contribution in the modelling of multi-agent systems. Existing multi-agent models in the RL literature have been built as refinement of single-agent MDPs, where agents take their decisions simultaneously at every step. The evolution of the environment state from one step to the next is driven by the joint action of all agents. This does not naturally describe many high-level multi-agent planning problems, in which agents can choose their next tasks at different times, as in VRPs. Starting from an MMDP, we will introduce a few properties of such planning processes that deal with abstract actions or tasks. It will lead us to a new variant of MMDP we call sequential MMDP (sMMDP). In this variant, agents do not take their decisions simultaneously anymore. Instead, only a single agent acts at each step, because actions naturally have a temporal dimension. With sequential individual decisions, the agent currently acting has access to more information, because it can observe the results of all the actions other agents have previously taken. The decision taken at each step is also much simpler, as it has to consider only the individual actions available to the current agent, instead of a full set of joint actions that grows exponentially with the number of agents. This invites us to treat the system as event-based, and trigger a decision for a particular agent only when it has finished its previous action, or on some changes of the environment state. If the

Multi-Agent RL (MARL) literature mainly focused on simultaneous decisions models inherited from MDPs, sequential models have been extensively studied in the Game Theory literature [58, 108]. The sMMDP model we propose can be seen as a special case of a more general collaborative sequential Markov Game [99, 59], where agents act one after the other in an order resulting from the dynamics of the system, sharing a common objective to maximize a cumulated reward.

In Section 4.1, we will introduce a toy S-VRP as a motivating example that will help us illustrate the properties of our sMMDP. Afterwards, Section 4.2 will describe the defining properties of the sMMDP, and analyze what they imply on the state value function associated with such a process. Finally, we illustrate the advantages of a sequential model for problems naturally described with sequential decisions in a small experiment using our motivating example in Section 4.3.

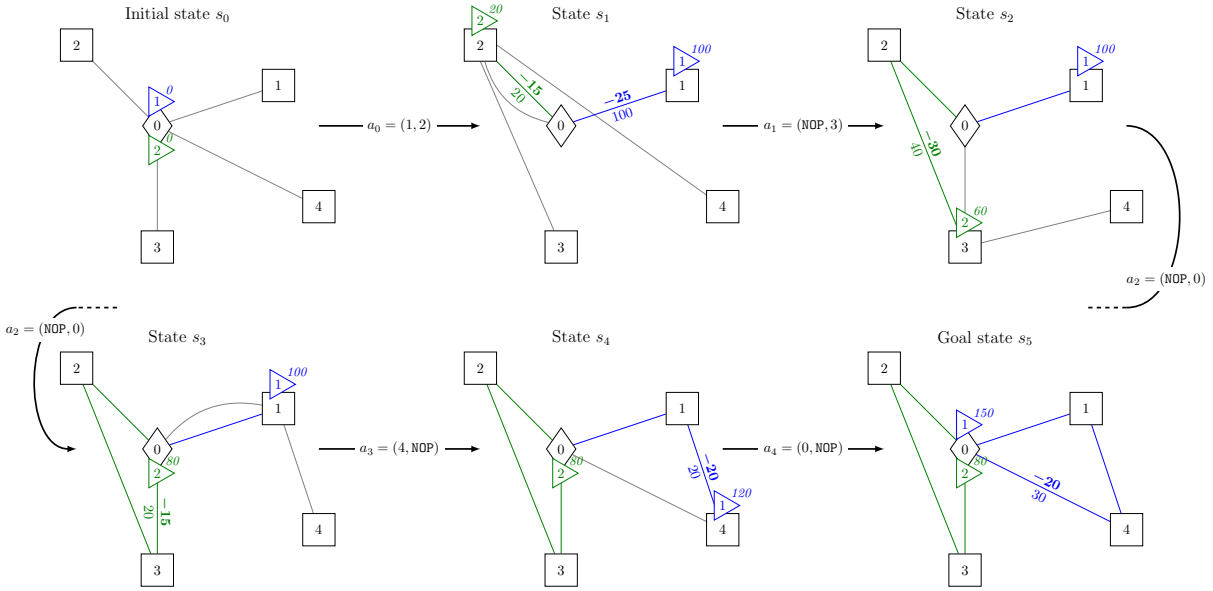
4.1 A toy S-VRP modeled as an MMDP

Before diving into our sequential variant of MMDPs, we introduce a motivating example inspired by the main focus in this thesis: routing problems. However, we will simplify it to focus on the structure and the properties we want to capture in our model. What we will describe in this section is a small VRP with stochastic travel times, and no additional operational constraints, for the sake of simplicity. After illustrating a trajectory sampled from the process in Subsection 4.1.1, we will informally explain how it can be modelled as a standard MMDP in Subsection 4.1.2. While doing so, we will highlight informally the properties we want to exploit in problems such as ours, and why a simultaneous model such as an MMDP is not well adapted to describe them. The processes we are interested in are indeed controlled by durative actions, or tasks. We could describe it more naturally using durative, sequential actions.

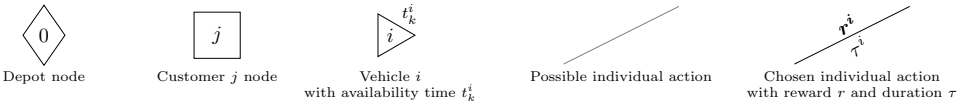
4.1.1 Motivating example

In this subsection, we will introduce the motivating example that will drive us through the entire section. We are interested in multi-agent processes with slow-paced decisions and uncertainty, such as concurrent tasks planning, or other high-level decision layers in multi-robot systems for example. The VRP, that is central in this thesis, is a particular case of such a problem. To keep it simple in this motivating example, we will get rid of all traditional operational constraints that come with such problems. For a more detailed formalization of a rich Dynamic and Stochastic VRP, we refer to Section 5.1 of the next chapter.

Example 4.1. Here, we consider $m = 2$ vehicles $\boxed{1}$ and $\boxed{2}$, both initially parked at a central depot $\diamondsuit{0}$. They have to travel to $n = 4$ customers $\boxed{1}$, $\boxed{2}$, $\boxed{3}$ and $\boxed{4}$, before returning to the depot. Every travel between two locations $j, j' \in \{0, 1, 2, 3, 4\}$ is associated with a cost $c(j, j')$, and takes some random duration $\tau \sim \mathcal{U}(\tau_{\min}, \tau_{\max})$ sampled from a discrete uniform distribution independently at each decision and revealed as soon as the vehicle leaves its previous location. Vehicles can choose a new destination only when they have reached their previous target. Once a customer is visited, it cannot be chosen as target anymore, and as soon as a vehicle travels back to the depot, it must stay there. A trajectory ends when both vehicles have returned to the depot. The goal of the vehicles is to find the routes that minimize the total travel cost.



(a) A complete trajectory in 6 steps in the toy S-VRP we use as motivating example.



(b) Legend

Figure 4.1 – Illustration of a trajectory in the small Stochastic VRP (S-VRP) that we use to illustrate the properties of our sMMDP model, built on top of the formalization of a standard MMDP. Two vehicles have to serve four customers. Travelling from one location to the next takes some time, and is associated with a cost. The goal is to find the optimal routes minimizing the total expected costs.

Figure 4.1 illustrates a small trajectory in this toy S-VRP, where vehicle 1 serves customer 1, while vehicle 2 has the time to serve 2 and 3. Afterwards, 2 returns to the depot 0 and 1 serves 4. Finally 1 also returns to 0. Now that we have setup this small problem, we will recall the formal notations of MMDPs and explain how we can use them to model it.

4.1.2 The initial MMDP model

We will start by modelling our small S-VRP described in Example 4.1 as a finite horizon goal-oriented MMDP [40]. In such a model, an episode starts from an initial state governed by the distribution P_0 and ends when reaching some goal states $s \in \mathcal{G}$. Given the definition of a basic MMDP (c.f. Definition 3.3 in Chapter 3), finite horizon goal-oriented MMDPs are formally defined as follow:

Definition 4.1 (Finite horizon goal-oriented MMDP)**.** A finite horizon goal-oriented MMDP is a Multi-agent MDP with finite horizon L and a subset of goal states $\mathcal{G} \subset \mathcal{S}$. Any goal state s in $\mathcal{G}$ is absorbing, meaning that once reached, no action will be able to cause a transition to any other state. Mathematically it translates to $P(s' \mid s, a) = \delta_s^{s'}$ and $R(s, a, s') = 0 \ \forall s \in \mathcal{G}$, where δ denotes the Kronecker delta, such that $\delta_y^x = 1$ if $x = y$, 0 otherwise. An extra reward $R^{\mathcal{G}}(s_L)$ is received at the end of the trajectory if the agents have reached a goal state, i.e. $s_L \in \mathcal{G}$.

Recall from Chapter 3 that our objective is to find a deterministic joint policy $\pi : \mathcal{S} \to \times_{i \in \mathcal{I}} \mathcal{A}^i$ indicating which individual action each agent has to take at any state in order to maximize the expected cumulated reward $\mathrm{E}[\sum_{k=0}^{L} R(S_k, A_k, S_{k+1})]$. As the state is fully observable for all agent, it is quite straightforward to extract individual policies π^i from this deterministic joint policy. Some solution methods based on policy gradients search the space of stochastic policies, in which it is non trivial to enforce that the optimal joint policy is separable into independent individual policies.

In our small S-VRP, the set of agents $\mathcal{I}$ is composed of the two vehicles 1 and 2. Without loss of generality, this set can be mapped to natural integers, so we will often write $i \in [\![1, m]\!]$ in the latter where $m = |\mathcal{I}|$ is the number of agents ($m = 2$ in this case).

A state $s \in \mathcal{S}$ of the system decomposes into multiple factors, namely the set of pending customers, the last customer served by each vehicle, and the time elapsed since each vehicle leaved the depot. For example at step 1 on Figure 4.1, customers 3 and 4 are pending, 1 is busy travelling to and serving 1 until $t_1^1 = 100$ minutes, and 2 has just finished serving 2 after $t_1^2 = 20$ minutes. A goal state $s \in \mathcal{G}$ is reached once both vehicles have returned to the depot 0 as we can see at step 5 on Figure 4.1. The set of goal states also includes a failure state $\texttt{FAIL} \in \mathcal{G}$ which prematurely ends a trajectory if vehicles choose an invalid action.

Each vehicle can choose from 6 possible individual actions in total: $a^i = 0$ to return to the depot, $a^i \in [\![1, 4]\!]$ to travel to and serve any of the 4 customers, or $a^i = \texttt{NOP}$ to do nothing and wait until it finishes serving its current target. However the two vehicles should not be able to serve a customer more than once, nor to interrupt a service they have committed to.

We enforce these constraints through the transition function P that triggers the failure state $\texttt{FAIL} \in \mathcal{G}$ as soon as one of them is violated. More importantly, P has what we call a *sequential* structure which can isolate the effect of each individual action on the different factors of the state. Here, it deterministically updates the vehicle's position and removes the chosen customer from the set of pending ones. The only stochastic factor comes from the travel duration which affects the cumulated time since departure of the vehicle, but it is also independent of the actions of the other agents.

The reward function R is given by the travel costs from the vehicles previous locations to their new locations. It can also be decomposed into additive / sequential terms, one per individual action. Additionally, we consider a completion reward, only added when the vehicles reach a goal state. We use it to penalize the vehicles for leaving customers pending at the end of their routes. At step 5 for example, 1 suffers costs $c(4, 0) = 20$ and 2 is idle (NOP) at depot with no cost. They reach a goal state and there is no customer left pending, so the reward is $R(s_4, a_4, s_5) = -c(4, 0) - 0 = -35$.

Finally, the initial state distribution P_0 is concentrated on a single state where both vehicles are located at 0 at time $t_0^1 = t_0^2 = 0$, with all customers pending. Concerning the planning horizon L, the maximum number of decisions the vehicles can make is $L = n + m = 4 + 2 = 6$ because they can only serve each customer once, and must come back to depot.

We have now completely modelled this simple motivating example as a goal-oriented MMDP. However while doing so, we highlighted that some properties of problems such as the small S-VRP we used as example are not well exploited by such a simultaneous model. It does not take full advantage of the separability of the transition and reward function, and has to explore the joint-action space which grows exponentially with the number of agents m, with a lot of dead

ends on fail state. Hence we propose a sequential variant of MMDP which would more naturally describe high-level multi-agent decision processes with slow-paced durative actions.

4.2 Building up a sequential MMDP

To better describe multi-agent systems where actions have different durations and are executed sequentially, we propose a new variant of MMDPs we call sequential MMDP (sMMDP). In this variant, agents do not take their actions simultaneously anymore. We take advantage of the separability of the transition and reward to make all individual actions sequential. Each decision step involves only a single agent. However, all agents still share a common objective and must cooperate to achieve it.

In this section, we will progressively introduce the properties we want to model, formalizing them in the case of a general (s)MMDP. We will illustrate each of them on our motivating example of Section 4.1.

4.2.1 Sequential states

The first property that we want to exploit is the structure of the states. We are indeed interested in multi-agent systems where we can isolate the effect of each individual action.

First let us recall what a state s is in a traditional MMDP: it is a statistic that contains all the necessary information to choose a joint action a and to describe how the system will evolve from one step to the next (respecting the Markov property). To take into account the effect of each individual actions separately, we define what we call *sequential states* as follow.

Definition 4.2 (Sequential states)**.** A sequential state, denoted $\bar{s} \in \bar{\mathcal{S}}$, is a statistic containing all the necessary information to:

1. indicate which agent has to act at a given time;
2. choose an **individual** action for this agent;
3. predict the next sequential state $\bar{s}'$ for the next agent;
4. estimate the immediate reward associated with any individual actions.

A decision process based on sequential states $\bar{\mathcal{S}}$ does not have the same temporal scale as its corresponding MMDP (see Figure 4.2). For every joint transition $s_k \to s_{k+1}$, there are m individual transition $\bar{s}_{m.k} \to \bar{s}_{m.k+1} \to \cdots \to \bar{s}_{m.k+m-1} \to \bar{s}_{m.k+m}$. For example in a two-agent system, $\bar{s}_3$ denotes the state reached in-between step 1 and 2 after applying agent 1 action.

It is important to note that the original state space $\mathcal{S}$ is included in this new sequential state set, i.e. $\mathcal{S} \subseteq \bar{\mathcal{S}}$. In particular, $\bar{s}_{m.k} = s_k$ is the state reached at step k before any agent actions is applied, and we loop back to $s_{k+1} = \bar{s}_{m.(k+1)}$ once we have applied action a_k^m of the last agent. For example in our toy S-VRP, when a joint action makes each vehicle serve a customer, we can go through an intermediate state where only one of the two customers has been served.

On top of this decomposition into sequential states, we can also decompose the state in individual and shared factors.

Definition 4.3 (Factorized states)**.** An MMDP is said to have factorized states if its state space $\mathcal{S}$ can be written as a Cartesian product:

$$\mathcal{S} = \mathcal{S}^0 \times \bigtimes_{i=1}^{m} \mathcal{S}^i \tag{4.1}$$

such that any state $s \in \mathcal{S}$ decomposes into factors $s = (s^0, s^1, s^2, \dots)$, where s^0 is a shared state factor controlled by all agents, and s^i is the individual state of agent $i \in \mathcal{I}$.

In our example the shared factor s^0 of the state corresponds to the set of pending customers. The individual factors s^1 and s^2 respectively regroup the position and time since departure of agents 1 and 2. This state structure will be key when designing approximate structure capable of representing it, either to estimate its value or derive decision rules.

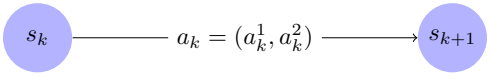
4.2.2 Sequential transitions

The second property which enables sequential MMDPs is the separability of the transition function. Because of this structure, we can isolate the effect of each individual action on the state of the whole system, as shown on Figure 4.2a and Figure 4.2b. Here, there are only two agents, and a transition from s_k to s_{k+1} caused by a joint action a_k decomposes into two sequential transitions from $\bar{s}_{2.k}$ to $\bar{s}_{2.k+1}$ because of a_k^1 then from $\bar{s}_{2.k+1}$ to $\bar{s}_{2.k+2}$ because of a_k^2. We will use this property to make individual decision step sequential at the end of this section.

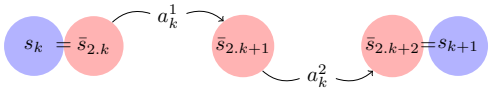
Definition 4.4 (Sequential transition)**.** An MMDP is said to have sequential transition if its transition function P can be written as a product

$$P(s_{k+1}|s_k, a_k) = P(\bar{s}_{m.(k+1)}|\bar{s}_{m.k}, a_k) = \prod_{i=1}^{m} P^i(\bar{s}_{m.k+i}|\bar{s}_{m.k+i-1}, a_k^i) \tag{4.2}$$

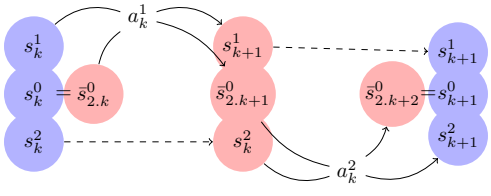
where P^i is a transition function indicating how the individual action a_k^i of agent i affects the global state of the system, going through all intermediate states $\bar{s}_{m.k+1}$ to $\bar{s}_{m.k+m-1}$ starting from $s_k = \bar{s}_{m.k}$ and finally reaching $\bar{s}_{m.(k+1)} = s_{k+1}$.



(a) Joint transition in the original MMDP



(b) Separable transition through an intermediate state $\bar{s}_{2.k+1}$



(c) Separable transition combined with factorized states

Figure 4.2 – Illustration of a transition from step s_k to step s_{k+1}, without factorized transition (a), with separable transition (b), and with separable transition and factorized states (c). States in blue are in the original state space $\mathcal{S}$ while states in red are in the intermediate state space $\bar{\mathcal{S}}$ accessible by individual actions.

In our example, we can apply independently the individual actions of both agents. When a vehicle chooses to serve a customer, it does not impact the service of the other vehicle. The corresponding partial transition function P^i removes the chosen customer from the set of pending ones, moves the vehicle to its new location and updates its time since departure.

Something interesting to notice in this example is that the structure of the separable transition can relate to the structure of the state. Typically, an individual action a_k^i will affect only the shared state s^0 and the individual state s^i of the agent i through the partial transition P^i, leaving the individual states $s^{i'}, i' \neq i$ of other agents untouched. We illustrate this combination of factorized transition and states on Figure 4.2c. Combining factorized and sequential states, we can write:

$$P(s_{k+1}|s_k, a_k) = \prod_{i=1}^{m} P^i(s_{k+1}^i, \bar{s}_{m.k+i}^0 | s_k^i, \bar{s}_{m.k+i-1}^0, a_k^i) \tag{4.3}$$

4.2.3 Sequential rewards

Next property that will be key to make the model sequential is the separability of the reward function.

Definition 4.5 (Sequential reward)**.** An MMDP is said to have a sequential (additive) reward if its reward function R can be written as a sum of sequential rewards:

$$R(s_k, a_k, s_{k+1}) = R(\bar{s}_{m.k}, a_k, \bar{s}_{m.(k+1)}) = \sum_{i=1}^{m} R^i(\bar{s}_{m.k+i-1}, a_k^i, \bar{s}_{m.k+i}) \tag{4.4}$$

where R^i are sequential reward functions scoring each individual action on every partial transition through intermediate states $\bar{s}_{m.k+i} \in \bar{\mathcal{S}}$.

Recalling our example, the sequential reward function R^i corresponds to the negative travel cost of each individual travel. The completion reward $R^{\mathcal{G}}$ penalizes the vehicles for leaving customers pending at the end of the trajectory.

It is important to notice that even if the immediate reward can be separated per actions, the goal of every agent is still to maximize the global cumulated reward collected. Said otherwise, an action is evaluated on all its consequences for all the agents, meaning we still consider a joint cooperative objective.

4.2.4 Sequential MMDPs

Using the structures we identified in previous subsections in terms of transition and reward functions, we are now ready to formally define what a sMMDP is.

Definition 4.6 (sequential MMDP)**.** A sequential MMDP is a Multi-agent MDP where agents take their decision sequentially. It is formally defined by the tuple $(\mathcal{I}, \bar{\mathcal{S}}, \mathcal{G}, \mathcal{A}^i, P^i, R^i, R^{\mathcal{G}}, P_0, L)$, where $\mathcal{I}$, $\bar{\mathcal{S}}$, $\mathcal{G}$ and $\mathcal{A}^i$ are respectively the sets of agents, intermediate states, goal states, and individual actions. P^i, R^i, $R^{\mathcal{G}}$ and P_0 are the transition, reward, goal reward, and initial state distribution functions, respectively. L is the (finite) planning horizon.

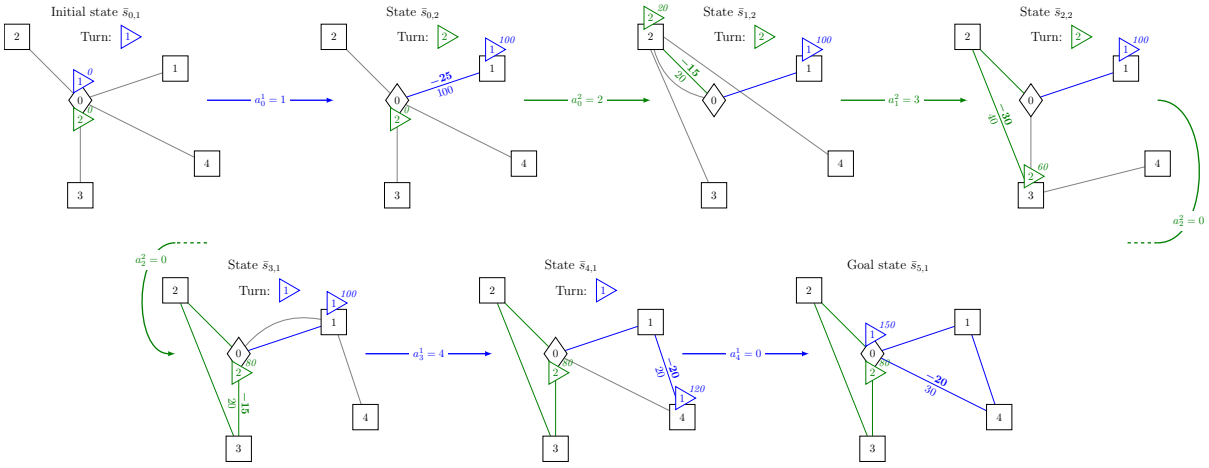


Figure 4.3 – The same motivating example of a small S-VRP but introducing sequential individual decisions. The next agent to act $i_k \in \{1, 2\}$ is the one whose current action finishes first. We can easily keep track of which agent is next using the cumulated time since departure. We skipped all intermediate states corresponding to a `NOP` action, which have no effect on the state.

If we reframed the motivating example previously described as a MMDP, we get the trajectory illustrated on Figure 4.3. Here, we can skip all the `NOP` actions of vehicles committed to a service and use sequential transitions and rewards to directly jump to sequential states where one of the vehicles has to make a decision. Because a single agent acts at each step in a sMMDP, we can define a turn function σ defined as follow:

Definition 4.7 (Turn function)**.** The turn function $\sigma : \bar{\mathcal{S}} \to \mathcal{I}$ indicates which agent $\sigma(\bar{s})$ has to act in any sequential state $\bar{s}$.

The goal is now to find a sequential policy $\pi_k : \bar{\mathcal{S}} \to \cup_{i \in \mathcal{I}} \mathcal{A}^i$ indicating the optimal individual action $\pi_k(\bar{s})$ for the agent making a decision on state $\bar{s}$ at step k to maximize the expected cumulated reward $\mathrm{E}[\sum_{k=0}^{L} \sum_{i=1}^{m} R^i(\bar{S}_{m.k+i-1}, A_k^i, \bar{S}_{m.k+i}) + R^{\mathcal{G}}(\bar{S}_{m.L})]$. Because the decisions are based on the sequential states $\bar{\mathcal{S}}$, it is much easier than in standard MMDPs to search for stochastic individual policies $\pi_k^i(a^i|\bar{s})$. Indeed, a stochastic sequential joint policy can be decomposed in terms of individual policies as $\pi_k(a^i|\bar{s}) = \pi^{\sigma}(\bar{s})_k(a^i|\bar{s})$, whereas in the simultaneous case, the separability of the joint policy needs to be constrained.

Similarly to classical MDP and MMDP, we can define the state value function V^{π} of policy π that indicates the expected cumulated rewards from any state $\bar{s}$ when agent i has to act, following joint policy π until reaching a goal state.

Definition 4.8 (Sequential state value)**.** In a sMMDP, the state-value function $V^{\pi}_{m.k+i-1}$ of a sequential policy π at step $k < L$ when agent i has to act gives the expected cumulated reward when following π from state $\bar{s}_{m.k+i-1}$, and onwards. Formally, it is defined as:

$$V^{\pi}_{m.k+i-1}(\bar{s}_{m.k+i-1}) = \mathrm{E}\left[\sum_{l=m.k+i-1}^{m.(L-1)} R^{\sigma(\bar{S}_l)}(\bar{S}_l, A_l^{\sigma(\bar{S}_l)}, \bar{S}_{l+1})\right] \tag{4.5}$$

Additionally, the value $V^{\pi}_{m.L}(\bar{s})$ when reaching the planning horizon L is given by the completion reward $R^{\mathcal{G}}(\bar{s})$.

Just as standard MDPs, the state value function V^{π} in sMMDPs obeys a Bellman recursive relation.

Theorem 4.1 (Sequential Bellman equation)**.** *In a sMMDP, the state-value function V^π of policy π obeys the following recursion:*

$$V^\pi_{m.k+i-1}(\bar{s}_{m.k+i-1}) = \mathrm{E}_{P^i,\pi}[R^i(\bar{s}_{m.k+i-1}, A^i_k, \bar{S}_{m.k+i}) + V^\pi_{m.k+i}(\bar{S}_{m.k+i})] \tag{4.6}$$

Proof. Starting from the definition of $V^\pi_{m.k+i-1}$ at intermediate sub-step $m.k+i-1$ and isolating the immediate reward, we can identify the remaining sum as $V^\pi_{m.k+i}$

$$\begin{aligned}
V^\pi_{m.k+i-1}(\bar{s}_{m.k+i-1}) &= \mathrm{E}\left[\sum_{l=m.k+i-1}^{m.(L-1)} R^{\sigma(\bar{S}_l)}(\bar{S}_l, A_l^{\sigma(\bar{S}_l)}, \bar{S}_{l+1})\right] \\
V^\pi_k(\bar{s}_{m.k+i-1}) &= \mathrm{E}\left[R^i(\bar{s}_{m.k+i-1}, A^i_k, \bar{S}_{m.k+i}) + \sum_{l=m.k+i}^{m.(L-1)} R^{\sigma(l)}(\bar{S}_l, A_l^{\sigma(l)}, \bar{S}_{l+1})\right] \\
V^\pi_k(\bar{s}_{m.k+i-1}) &= \mathrm{E}[R^i(\bar{s}_{m.k+i-1}, A^i_k, \bar{S}_{m.k+i}) + V^\pi_{m.k+i}(\bar{S}_{m.k+i})]
\end{aligned}$$

□

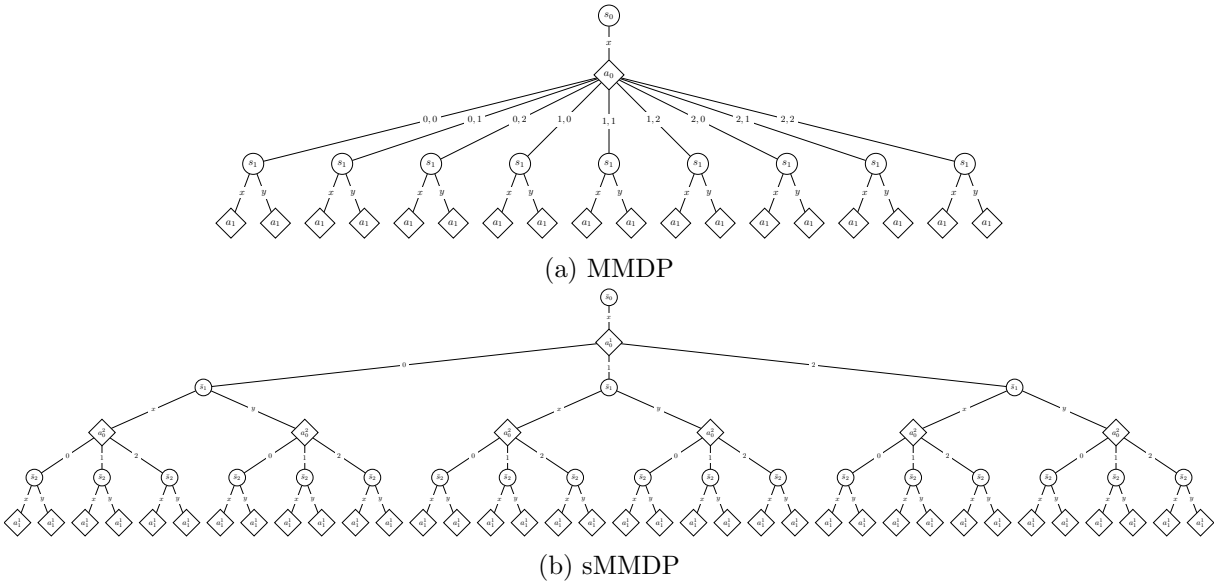
From this property, we can deduce the following Bellman optimality equation.

Theorem 4.2 (Sequential Bellman optimality equation)**.** *The optimal state-value function V^* for a sMMDP can be recursively defined as follow:*

$$V^*_{m.k+i-1}(\bar{s}_{m.k+i-1}) = \max_{a^i \in \mathcal{A}^i} \mathrm{E}\left[R^i(\bar{s}_{m.k+i-1}, a^i, \bar{S}_{m.k+i}) + V^*_{m.k+i}(\bar{S}_{m.k+i})\right] \tag{4.7}$$

This properties follows from the sequential Bellman equation of Theorem 4.1 and can be proven using the same steps as for traditional MDP. Indeed a sMMDP can be viewed as an equivalent centralized MDP with augmented state $\bar{\mathcal{S}}$. The Bellman optimality equation shows that the sequentiality of sMMDP preserves the optimality of deterministic policies, as the individual policies defined by taking the arg max in Equation 4.7 are optimal. Parameterized stochastic policies can still be useful to search in a continuous solution space using Stochastic Gradient Descent (SGD).

If we compare the complexity of sMMDPs to the one of MMDPs, we can see that we made a trade-off between the augmented state-space dimension and the time-horizon which grows by a factor m one side, and the exponential simplification of the joint action-space to the individual action-space of each agent. To illustrate this, let us consider a sMMDP with 2 agents, 2 states, and 3 individual actions, and the corresponding MMDP which then has $3^2 = 9$ joint actions. Figure 4.4 depicts the decision tree on the first decision step of both models. We can see that the number of decision nodes stays in the same order of magnitude in both case. Indeed the sMMDP has an exponentially lower branching factor and a linearly greater depth than its corresponding MMDP.



(a) MMDP

(b) sMMDP

Figure 4.4 – Comparison of decision trees for the MMDP and sMMDP models.

4.3 Simultaneous vs. sequential model

To test the performances of our sequential model compared to the original simultaneous one, we implement two tabular Q-Learning algorithms to solve our motivating example. The first one uses simultaneous joint action and the initial *MMDP* model (Algorithm 4.1). The second one (Algorithm 4.2) uses sequential individual actions and the *sMMDP* model. We randomly sample the costs $c(j, j')$ and the travel times τ from uniform distributions using a random number generator initialized with different fixed seeds.

```
for a fixed number of iterations do
    s ← reset_environment();
    while s not a goal state do
        if random() < ϵ then
            a ← random_joint_action();
        else
            a ← arg max_a Q[s, a];
        s, r ← step_environment(a);
        v′ ← max_a′ Q[s′, a′];
        Q[s, a] ←
          (1 − α)Q[s, a] + α(r + v′);
        s ← s′;
    ϵ, α ← update_metaparameters();
```

Algorithm 4.1: Q-Learning

```
for a fixed number of iterations do
    s̄, i ← reset_environment();
    while s̄ not a goal state do
        if random() < ϵ then
            a^i ← random_indiv_action();
        else
            a^i ← arg max_{a^i} Q[s̄, i, a^i];
        s̄′, i′, r ← step_environment(a^i);
        v′ ← max_{a′^i} Q[s̄′, i′, a′^i];
        Q[s̄, i, a^i] ←
          (1 − α)Q[s̄, i, a^i] + α(r + v′);
        s̄, i ← s̄′, i′;
    ϵ, α ← update_metaparameters();
```

Algorithm 4.2: Sequential Q-Learning

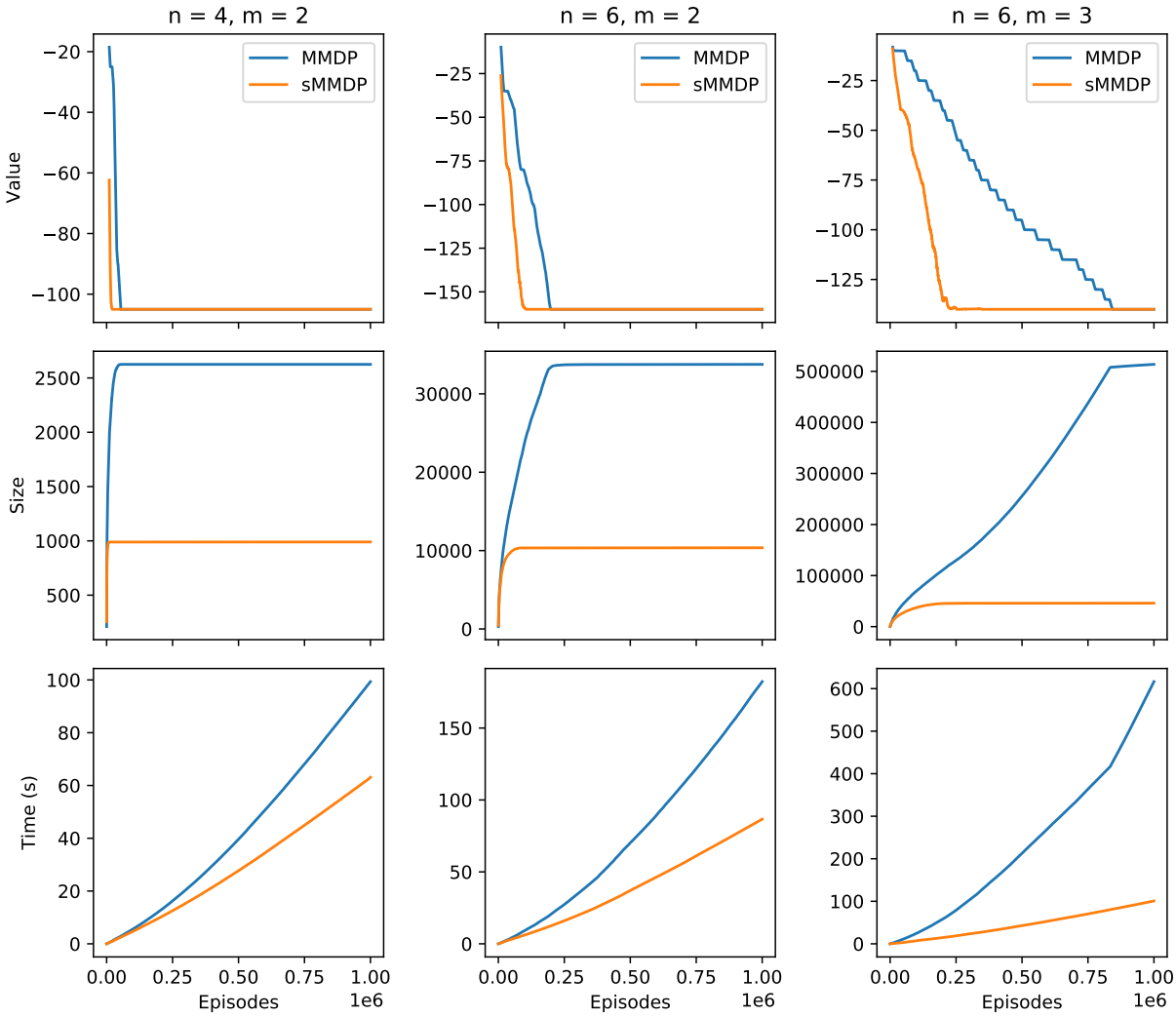


Figure 4.5 – Learning curves of tabular Q-Learning applied to the MMDP and sMMDP models of our toy S-VRP with different numbers of customers n and vehicles m, in terms of objective value, size of the Q-value table and execution time.

We slightly varied the number of customers and the number of vehicles to test how both approaches scale up. However, the tabular Q-value representation quickly grows towards intractable sizes, so in this section, we limit our experiments to three setups `ToySVRP(4,2)`, `ToySVRP(6,2)` and `ToySVRP(6,3)` where the first number is the number of customers ($n = 4$ or 6) and the second one ($m = 2$ or 3) is the number of vehicles. The approximation structure we use to address larger and richer DS-VRPs is detailed in Chapter 5.

We used the same set of meta-parameters (number of episodes, learning rate and epsilon greedy exploration) for both *MMDP* and *sMMDP* approaches. Experimental results in terms of objective value, size (number of entries in the Q-value table), and execution time are reported on the learning curves of Figure 4.5. As we can see, both approaches converge in the given number of iterations in our small problems. The sequential model enables us to drastically reduce the size of the Q-value tables, and iterations applying the Bellman update operator are much faster than in the standard MMDP. Scaling with the dimensions of the problems in terms of size and time is also clearly in favor of the sMMDP in that case.

4.4 Model discussion

Constrained actions subset. Many high-level planning problems such as the ones we are interested in have to respect some hard constraints on the actions that the agents can take at each step. In the classical RL literature, impossible actions would simply make the system transition to a sink state with highly negative rewards. However, in our cases, these actions are easily identifiable from the state of the system, and we would like to avoid using many training samples just to learn to avoid choosing these actions at the wrong time. Also, when the problems grows and the number of states and actions becomes intractable, we have to use approximation structures to represent our value functions or our policies. Some of these approximation structures, including Deep Neural Networks (DNNs), rely on (learned) kernels that project the states to some internal representation space. The discontinuities caused by invalid actions in the reward signal can be very hard to isolate in these internal spaces, and bias the approximation, reducing its performances. In next chapter, we will develop how we encode the set of possible actions and match it against the internal state of the agent to create a decision rule that only consider valid actions.

To integrate this property in our model, we consider that there exists a known, deterministic function $\Xi^i : \bar{\mathcal{S}} \to \wp(\mathcal{A}^i)$ that yields the subset of valid actions $\Xi^i(\bar{s}) \subseteq \mathcal{A}^i$ given the global state $\bar{s}$ of the system, where $\wp(\mathcal{A}^i)$ denotes the power-set, i.e. the set of all possible subsets, of the set of individual actions $\mathcal{A}^i$. In our example, agents (i.e. vehicles) can only travel to pending customers and cannot choose the same target, except if they both return to the depot. It is much easier to express this constraint in the individual action space based on the intermediate state resulting from the separated execution of individual actions, instead of in the joint action space.

Temporally extended actions. Another property that is at the core of the problems we want to address is the slow pace of the decisions. Contrary to lower-level control systems which need to work at high frequencies, our abstract decisions naturally describe some long-lasting actions. Such temporally extended actions have been studied in Semi-Markov Decision Processes (SMDPs) [17, 63]. This model explicitly considers stochastic continuous time in-between decisions during which a continuous reward is perceived. Additionally, the options framework [107] builds a bridge between MDPs and SMDPs, taking benefit of an underlying MDP model to build a hierarchical temporal abstraction of decisions. Multi-agent extension of SMDPs or options are still rare [76]. In our sMMDP model, we aim at taking advantage of the problem structure and actions durations to simplify the agents interactions through time.

However, contrary to models applying the options framework to multi-agent systems, we simplify the state space we have to explore by considering our transition function can directly predict the final result of each individual actions. We still preserve the order in which the agents act by also predicting the completion time of all actions. In this configuration, the sMMDP model becomes much more interesting, because most of the intermediate actions will be `NOP` while waiting for the initial action termination. In these conditions, the sMMDP has a net advantage over the simultaneous MMDP approaches. Indeed it can skip all intermediate steps corresponding to agents that must apply a `NOP` action in the current state. The function σ can be repurposed as a turn function $\bar{\mathcal{S}} \to \mathcal{I}$ which indicates based on the state which agents has to act. In the general case, it could even be a stochastic function $\bar{\mathcal{S}} \times \mathcal{I} \to [0, 1]$, as long as the sequential states contains enough information to predict which agent has to act at any state.

4.5 Conclusion

In this chapter, we discussed how to model slow-paced decision processes involving multiple agents. These agents interact with a common environment through temporally extended actions whose effects on the state can be isolated. Thanks to this property, we defined a variant of MMDP called sequential MMDP (sMMDP) which describes decision processes where agents do not act simultaneously anymore. Instead, only one agent acts at a given step, causing a state transition and receiving a reward. However, because a factor of the state is shared between agents, and because they pursue a common objective, the problem cannot be solved independently. There are also constraints on the available individual actions which depend on the shared factor of state, which interleave the trajectory of each agent.

sMMDPs are better suited to describe problems that naturally involve sequential decisions. The agents have access to more information to take their decisions at each step. Also, while having only one agent make a decision at each step causes the trajectory to be longer, it involves searching only in the individual action space instead of the exponentially larger joint action space. That is why we can preserve the complexity of the problem we try to model. Even with the reduction of the number of entries in the Q-value table we observed in our experiments, larger problem are still intractable using such an exhaustive representation. To solve this issue, we will now propose an approximation architecture which relies on the state structures identified in our sMMDP model to extract features from the state of the system and output decision rules online for any agent, taking advantage of the properties of the model. It enables us to extend the discussion on efficient state representations we started in Chapter 3.

We will now use the models we introduced here to address our initial problem: a rich Dynamic and Stochastic VRP. More specifically, we will learn a parameterized policy implemented as an architecture we call Multi-agent Routing Deep Attention Mechanisms (MARDAM) that generalizes to unknown instances of the problem, with different sets of customers configurations.

Chapter 5

Multi-Agent Routing using Deep Attention Mechanisms

After describing standard models and solution methods for Vehicle Routing Problems (VRPs) in Chapter 2, we introduced in Chapter 3 Markov Decision Processes (MDPs) borrowed from the Reinforcement Learning (RL) literature, that are naturally well suited to describe Dynamic and Stochastic variants of the problem (DS-VRPs). We analyzed the structure and properties of our problems of interest in Chapter 4, which led us to define a new variant of Multi-agent MDP we call sequential MMDP (sMMDP). We conducted an experiment comparing sMMDP to its MMDP counterpart. One of the conclusion of this experiment was that even though sMMDP behaved better in term of scaling up the number of vehicles and customers, an exhaustive representation of the value function is still intractable for larger problems. Hence, we need an approximation structure that can efficiently capture the properties of the problem.

We now have all the tools we need to address rich DS-VRPs. In this chapter, we will first provide details about how we model DS-VRPs with Capacitated vehicles, stochastic travel times and stochastic customers with Time Windows (DS-CVRPTWs) as a sMMDP. To circumvent the issue of scaling up the dimensions of the problems, adapted approximation structures are required. To learn to represent the complex state of rich DS-VRPs, we introduce a new policy architecture we call Multi-Agent Routing Deep Attention Mechanisms (MARDAM) network. It is designed around specialized encoders for the customer configuration, the vehicles states (in continuous space) and a flexible scoring of possible next travels (actions). Unfortunately, we then cannot guarantee the optimality of the solution, nor give bounds on its optimality gap. The best we can say is that we will find a local optimum in the class of function we are capable of representing. However, because this class is very rich when using DNNs as approximation structures, we expect our approach to output high quality solutions. We will empirically evaluate if this claim holds in Chapter 6. We published these model, architecture and the experimental results that accompany them in [1].

5.1 Modelling a DS-CVRPTW as a sMMDP

The first step before we can develop on our approach using Attention Mechanisms on DS-VRPs is to model the problem as a sMMDP. It will enable us to train a policy that can take online decisions depending on the evolution of the available information. Recall from Definition 4.6 that a sMMDP is formally defined by the tuple $(\mathcal{I}, \bar{\mathcal{S}}, \mathcal{G}, \mathcal{A}^i, P^i, \sigma, R^i, R^{\mathcal{G}}, P_0, L)$. The following

subsections give details on each of these elements. We provide an extensive formalism in order to show how the model will be fully integrated in a Deep RL training loop, using efficient tensor structures and operations optimized in many standard frameworks.

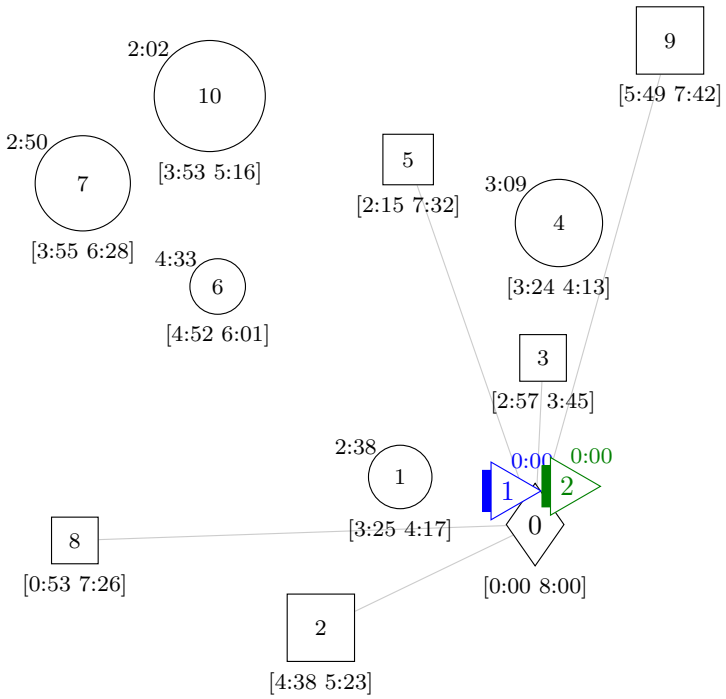


Figure 5.1 – Example of a DS-CVRPTW with 10 customers that we will model as a sMMDP. It is only a sample of all possible customer configurations, which are randomly distributed. ◇0 represents the depot where all vehicles are initially parked. □j are the customers initially known, and ○j are dynamically appearing customers. The node sizes are proportional to their demands, and their time window $[e^j, l^j]$ is indicated below. Service durations are not represented. The appearance time b^j of dynamic customers is indicated at the top left. Vehicles ▷1 and ▷2 start at depot. Their remaining capacities are shown as filled rectangles, and their availability times as a label above them.

Figure 5.1 illustrates a sample of a DS-CVRPTW with 10 customers, 5 of which are dynamic customers, i.e. they appear only after some time. Their appearance time, and all their other features are not known/observable a priori. The two vehicles ▷1 and ▷2 start at the depot at time $t_0^1 = t_0^2 = 0$, with maximum capacity $\kappa_0^1 = \kappa_0^2 = 50$.

5.1.1 Agents set

Each vehicle composing the fleet is an autonomous agent $i \in \mathcal{I}$ in the system. As introduced before in Chapter 2, we call $m \in \mathbb{N}$ the number of vehicles in the fleet, hence $\mathcal{I} = [\![1, m]\!]$. In the latter, we will consider problems with up to $m = 10$ vehicles, but here in our example of Figure 5.1, there are only $m = 2$ vehicles. However, we will benefit from the symmetries there are between the different agents when the fleet is homogeneous to share experiences between them and improve the efficiency of our learning procedure. Some more traditional OR heuristic approaches can handle problems with a much higher number of vehicles, up to hundreds, or even thousands of vehicles [100]. The limiting factor for these methods is the number of customers,

which is also higher than what we are capable of representing with DNN. However most standard OR approaches rely on hand-crafted expert heuristics, whereas DNN opens up possibility to learn such heuristics, adapting to past experiences while preserving the capacity to generalize to unknown problems.

5.1.2 State space

As previously introduced in Section 4.2 of the last chapter, any state $\bar{s} \in \bar{\mathcal{S}}$ is factorized into a shared state $\bar{s}^0$ and individual states $\bar{s}^i$ for every agent $i \in [\![1, m]\!]$.

Shared customer state The shared state $\bar{s}^0$ not only indicates the set of pending customers, represented as a boolean vector $\xi \in \{0,1\}^{n+1}$, such as $\xi_j = 1$ if customer j is still pending and $\xi_j = 0$ if j has already been served. $\bar{s}^0$ also contains their configuration. Indeed customers coordinates (x^j, y^j), demands q^j, TWs $[e^j, l^j]$, durations d^j and appearances b^j are sampled when initializing the trajectory. Formally, we can write $\bar{s}^0$ as a matrix of size $(n+1) \times 8$ where each row contains the features of customer j, and a boolean ξ^j indicating whether customer j is pending or not (depot is always "pending"), with n the number of customers.

$$\bar{s}^0 = \begin{bmatrix} x^0 & y^0 & 0 & 0 & l^0 & 0 & 0 & 1 \\ x^1 & y^1 & q^1 & e^1 & l^1 & d^1 & b^1 & \xi^1 \\ & & & \dots & & & & \\ x^j & y^j & q^j & e^j & l^j & d^j & b^j & \xi^j \\ & & & \dots & & & & \\ x^n & y^n & q^n & e^n & l^n & d^n & b^n & \xi^n \end{bmatrix} \quad (5.1)$$

State $\bar{s}^{0,j}$ of customer $j \in [\![0, n]\!]$

Demands of all customers

Pending status of all customers

Because the number of customers can vary, the shared state encoder block of the policy, also called customers encoder, will need an efficient way to represent sets of vectors. See the following Section 5.2 for the solution we selected to implement this customer encoder. That makes our problem a VRP with stochastic customers, adding a first source of uncertainty.

For example, the features of the customers in the instance illustrated in Figure 5.1 at time $t = 0$ where the dynamic customers have not appeared are described by the matrix:

$$\bar{s}_0^0 = \begin{bmatrix} 32 & 20 & 0 & 0 & 480 & 0 & 0 & 1 \\ 5 & 33 & 12 & 278 & 323 & 15 & 0 & 1 \\ 33 & 1 & 6 & 177 & 225 & 17 & 0 & 1 \\ 16 & 26 & 9 & 135 & 452 & 26 & 0 & 1 \\ -26 & -22 & 6 & 53 & 446 & 34 & 0 & 1 \\ 49 & 41 & 12 & 349 & 462 & 12 & 0 & 1 \end{bmatrix} \quad (5.2)$$

If we consider that vehicles 1 and 2 have respectively served customers 8 and 5, enough time has gone by for customers 1 and 10 to appear. The customers state $\bar{s}_2^0$ is then:

$$\bar{s}_2^0 = \begin{bmatrix} 32 & 20 & 0 & 0 & 480 & 0 & 0 & 1 \\ 15 & -14 & 8 & 205 & 257 & 18 & 158 & 1 \\ 5 & 33 & 12 & 278 & 323 & 15 & 0 & 1 \\ 33 & 1 & 6 & 177 & 225 & 17 & 0 & 1 \\ 16 & 26 & 9 & 135 & 452 & 26 & 0 & 0 \\ -26 & -22 & 6 & 53 & 446 & 34 & 0 & 0 \\ 49 & 41 & 12 & 349 & 462 & 12 & 0 & 1 \\ -9 & 34 & 14 & 233 & 316 & 24 & 122 & 1 \end{bmatrix} \quad (5.3)$$

1 and 10 have appeared.

5 and 8 have been served.

Individual vehicle state Each agent state $\bar{s}^i$ is a vector composed of the coordinates (x^i, y^i) of the last customer it served, its remaining capacity κ^i and its next availability time t^i (corresponding to the cumulated time since departure, including travel, wait and service times).

$$\bar{s}^i = \begin{bmatrix} x^i & y^i & \kappa^i & t^i \end{bmatrix} \quad (5.4)$$

The two last components describe the same dynamic quantities as the state variables of Equations (2.2) and (2.3) in the MILP formulation, but they are associated with a vehicle instead of a node (a customer). That is why we chose to introduce the MILP formulation of Cordeau et al.[24], which uses quadratic/conditional constraints to maintain the consistency of state variables for capacity and time.

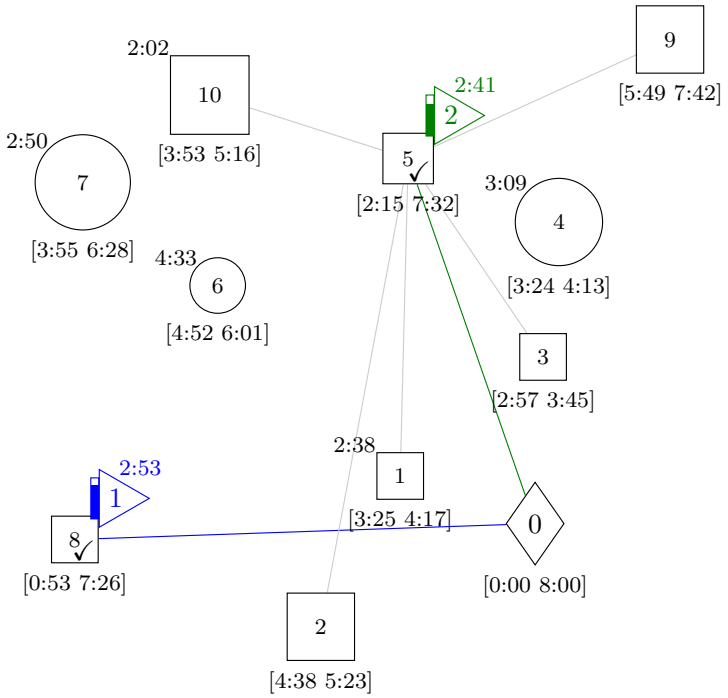


Figure 5.2 – State reached after 1 decided to serve 8 and 2 served 5. Vehicles have moved, their capacities and availability times have been updated, served customers are marked as such, and customers 1 and 10 have appeared.

Taking the same example trajectory as in previous paragraph, where 1 and 2 have served 8 and 5, we reach the state $\bar{s}_2$ depicted in Figure 5.2. At this step, we have:

$$\begin{array}{rcl} \bar{s}_2^1 & = & [-26 \quad -22 \quad 44 \quad 173] \\ \bar{s}_2^2 & = & [16 \quad 26 \quad 41 \quad 161] \end{array} \tag{5.5}$$

Goal states A goal state $\bar{s} \in \mathcal{G}$ is reached when all vehicles are back to the depot. Such a goal state is illustrated on Figure 5.3.

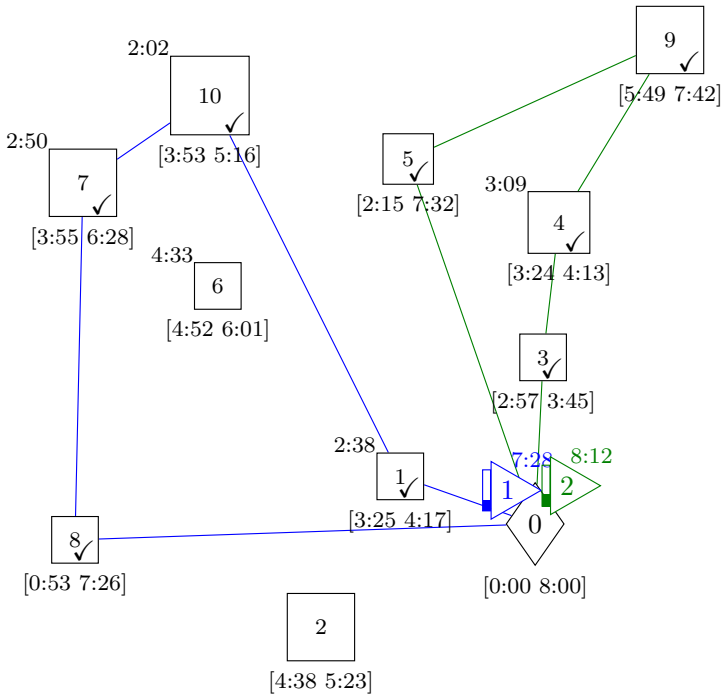


Figure 5.3 – Example of a goal state reached after 1 served 8, 7, 10, and 1, while 2 served 5, 9, 4, and 3.

5.1.3 Constrained individual actions

In this formalization of DS-VRPs as sMMDPs, an individual action consists in travelling to and serving a customer or return to the depot, It means that the set of all possible individual actions can be mapped to $\mathcal{A}^i = [\![0, n]\!]$ for all agents $i \in [\![1, m]\!]$, where n is the number of customers. $a^i = 0$ corresponds to returning to the depot 0, and $a^i = j$ to serving customer j.

The constraints of the DS-VRP can be expressed as a partition between valid and invalid actions. An action $a^i = j$ can only be selected by a vehicle i at step k if:

1. j is still pending, i.e. $\xi_k^j = 1$ in the shared state matrix $\bar{s}_k^0$
2. j has already appeared when i takes its decision, i.e. $t_k^i \geqslant b^j$
3. i has enough remaining capacity to serve j, i.e. $\kappa_k^i \geqslant q^j$

Given these three constraints, we can define the set of possible actions for i at step k as a deterministic function of the state denoted $\Xi^i : \bar{\mathcal{S}} \to \{0,1\}^{n+1}$, such as $\Xi^i(\bar{s})$ is a binary vector whose components are 1 only where the three conditions stated above are met. Checking the validity of each action can be made for all customers and vehicles at once, through element-wise operations on vectors extracted from the shared and individual states. These binary vector will be later used as an input of our policy network MARDAM (c.f. Section 5.2), to enforce that invalid actions have no probability to be selected in the decision rules it outputs.

For example, if we consider the state $\bar{s}_2$ depicted on Figure 5.2, the valid actions for 2 are given by the binary vector:

$$\Xi^2(\bar{s}_2) = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{5.6}$$

Here, the remaining capacity κ_2^2 of 2 is never a limiting factor, and all pending customers can be served.

5.1.4 Turn and transition functions

Now that we have identified the structures we use to describe all possible states $\bar{\mathcal{S}}$ of the DS-CVRPTW, and how we classify actions of travelling to customers as valid or invalid, we will define how these states evolve given the action of the vehicles. As we want to model a sequential system, we only have one vehicle acting at a time, and we will define how the information of which vehicle acts next can be extracted from the state.

Transition function The transition function P^i is for the most part deterministic. When agent i travels to customers j after choosing action $a_k^i = j$, its individual state $\bar{s}_k^i$ gets updated as followed:

$$\bar{s}_{k+1}^i = \begin{bmatrix} x^j & y^j & \kappa_k^i - q^j & \max\{t_k^i + \tau^{ij}, e^j\} + d^j \end{bmatrix} \tag{5.7}$$

where $\tau^{ij} = \frac{d^{ij}}{\mathcal{V}_k^i}$ is the time it takes to vehicle i to travel from its previous position (x_k^i, y_k^i) to the position of its targeted customer (x^j, y^j). In our setup, this travel time is a random variable resulting from the euclidean distance d^{ij} between the two locations and the stochastic travelling speed $\mathcal{V}_k^i$ of the vehicle at this step.

This speed is sampled independently for each travel from a continuous bimodal distribution: the first mode emulates flowing traffic with nominal vehicle speed V_{nom} and low variance σ_{nom}^2, while the second mode corresponds to congested traffic with reduced mean speed $V_{\text{slow}} < V_{\text{nom}}$ and higher variance $\sigma_{\text{slow}}^2 > \sigma_{\text{nom}}^2$. The following equation formalizes this distribution:

$$\begin{aligned} Pr\{\mathcal{V}_k^i = v\} = (1 - p_{\text{slow}})\ &\mathcal{N}(v; V_{\text{nom}}, \sigma_{\text{nom}}^2) \\ + p_{\text{slow}}\ &\mathcal{N}(v; V_{\text{slow}}, \sigma_{\text{slow}}^2) \end{aligned}$$

where $\mathcal{N}$ denotes the normal distribution. The resulting distribution for varying values of p_{slow} is illustrated on Figure 5.4. In this model, travel times are completely independent from one another. We consider that there is enough background traffic formed by the flow of uncontrolled vehicles such that the travel of one agent does not affect the ones of other agents. A finer model would need to consider the correlations between travel times given the road network, and the traffic conditions at every time of day.

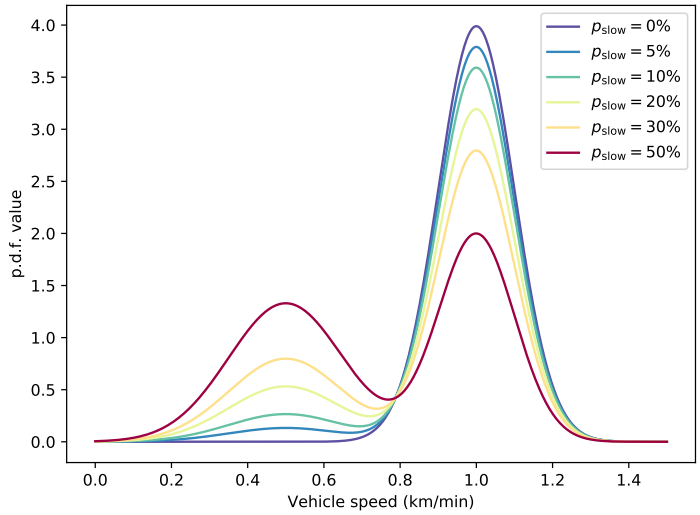


Figure 5.4 – Vehicle speed distribution for different slowdown probabilities

Shared state update The features of the customers stored in the first seven columns of the shared state $\bar{s}^0$ are sampled once at the beginning of the episode, but are then invariant along the trajectory. However the last column ξ_k is updated to indicate that customer a_k^i has just been served and is not pending anymore. We also check that no new customers have appeared between the last and current decision time.

Turn function Recall form Chapter 4 that in our sMMDP model, only one agent acts at a time. Any sequential state $\bar{s} \in \bar{\mathcal{S}}$ contains enough information to predict which agent has to act. We defined the turn function $\sigma : \bar{\mathcal{S}} \to \mathcal{I}$ which extract this information from the state, and return the agents currently awaiting a decision. In our model of DS-VRPs, it returns the agent with the smallest availability time $\sigma(\bar{s}_k) = \arg\min_{i \in [\![1,m]\!]} t_k^i$. Because the travel times are stochastic, the order in which the agents act is not pre-determined, even though the turn function is deterministic.

5.1.5 Reward function

The next element to define our DS-CVRPTW model is the reward function. It is the key element that will guide the optimization of the policy. That is why we will discuss the design of the reward signal, to match the objective of traditional DS-VRPs formulations, while trying to minimize the risk ill-defined solutions corresponding to local optima of the expected cumulated reward objective function in our RL framework.

The reward associated with each individual travel corresponds to the negative cost of travel in the original problem formulation. In our DS-CVRPTW, we use the euclidean distance d^{ij} between the coordinates of the customers, and we want to minimize the total distance travelled. In more realistic scenarios, it could be based on the fuel and maintenance costs of the vehicles. As we are considering soft TWs, a second term corresponding to a lateness penalty is subtracted to this base travel cost if the vehicle arrives to serve a customer after its due time. Formally, we write it as:

$$R^i(\bar{s}_k, a_k^i, \bar{s}_{k+1}) = -d^{ij} - C_{\text{late}} \max\{0, t_k^i + \tau^{ij} - l^j\} \tag{5.8}$$

The proportionality coefficient C_{late} associated with this penalty is a meta-parameter we can tweak to balance how late we are allowed to serve a customer. If we increase it enough, the lateness penalty can become a predominant term in the reward, so that we can mimic hard TWs. There is still one very important constraint of the CVRP we did not address in our model yet. If we can limit the actions available to an agent at any time such that no customers can be served twice, we still have no incentive to even serve any customer at all. The trivial optimal solution for our model at this stage is to stay at the depot indefinitely to avoid all costs. We considered multiple alternatives to solve this issue:

1. Give a positive reward to the agent when it successfully delivers its item to a customer compensating for the travel cost;

2. Associate the action of staying at the depot with an additional negative reward and only end the trajectory when all customers have been served;

3. Give a discouraging negative reward to the whole fleet at the end of the trajectory for any customer left pending.

The first alternative is maybe the most intuitive as it directly echoes the real economic incentives logistics companies are subject to. However, in reinforcement learning it is very hard to balance between a high enough value to avoid the trivial solutions of preferring the depot to any delivery, and a low enough value so that it does not completely overshadow the costs. Theoretically a high value should work, but in practice, most RL algorithms will fall into a local optima where every customer is served before it can even try to optimize for travel and lateness cost.

The second alternative is directly inspired by classical approaches to shortest path problems. As every action is associated with a cost, trajectories with as few decision steps as possible are emphasized. In our case any useless stay at the depot will only worsen the cost of the trajectory. However this alternative causes some problems in our sequential durative action model. When should a decision for a vehicle parked at the depot be triggered? Obviously at the beginning of the trajectory, all vehicles have to choose if they leave the depot or not. But after that, any "wait at depot" action could be extended indefinitely. If we triggered a new decision any time another vehicle finishes its action, we would loose on the advantages of considering durative actions to limit the size of the spaces to explore. One possibility is to consider only external events modifying the environment state, such as when a new client appears. Again, the value of this cost of waiting needs to be balanced so that vehicles are encouraged to serve the customers despite the travel cost, but also such that no extra vehicle is deployed when only part of the fleet can manage all the deliveries in time.

The last alternative is quite similar to the second one, as it also only involves negative rewards, which needs to be just high enough to discourage all vehicles to stay at the depot without serving any customer. However it has the advantage to not penalize vehicles that choose to stay at the depot when only part of the fleet can deal with all deliveries. That is why we converged to this reward model for our sMMDP, and integrated a completion reward $R^{\mathcal{G}}$ to the model.

$$R^{\mathcal{G}}(\bar{s}_{k+1}) = \begin{cases} -C_{\text{pend}} \sum_{j=1}^{m} \xi_{k+1}^{j} & \text{if } \bar{s}_{k+1} \in \mathcal{G} \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

To conclude on these thoughts on reward signal design, it is both a critical and a difficult part of any RL-based approach, which we adapted to our new formalization taking into account durative actions and sequential event-based decision.

5.1.6 Initial state distribution and planning horizon

As we quickly mention when describing the factored state space, the initial state distribution P_0 samples a set of customers with varying configurations. A detailed example of this sampling will be presented in Section 6.2 when discussing the benchmarks we used in our experiments. An example of a resulting instance of the problem is illustrated above in Figure 5.1. This could be seen as a generalization of a S-VRP with stochastic customers. The goal here is not only to adapt to the potential presence of some pre-defined customers but to train our learning agents to adapt to many different situations and use the same pre-trained policy for a variety of problems.

It also deterministically initializes every vehicle at depot with full capacity, which we can write:

$$\bar{s}_0^i = \begin{bmatrix} x^0 & y^0 & \kappa_{\max} & 0 \end{bmatrix} \qquad \forall i \in \mathcal{I} \tag{5.10}$$

where (x^0, y^0) are the coordinates of the depot, and $\kappa_{\max}$ is the full capacity of every vehicle. Finally, all customers are marked as pending in the vector ξ, i.e. :

$$\xi_0^j = \begin{cases} 1 & \text{if } b^j = 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

Concerning the horizon L, as vehicles cannot serve every customer more than once and must come back to depot, the total number of decision cannot exceed $L = n + m$.

5.2 MARDAM, a modular Policy Network for DS-VRPs

We have now described how we model a DS-CVRPTW as a sMMDP. As the problems we consider involve up to ten vehicles and fifty customers, the sizes of the state and action spaces become intractable, and we cannot use an exhaustive representation of the policy. To circumvent this issue, adapted approximation structures are required. We are only guaranteed to find a local optimum in the class of function we are capable of representing. However, because this class is very rich when using DNNs as approximation structures, we expect our approach to output high quality solutions. That is why we propose a modular DNN architecture to implement a parameterized stochastic policy which we will train using an Actor-Critic algorithm. We call this specialized policy network *Multi-agent Routing Deep Attention Mechanisms*, or MARDAM. In this section, we will detail how the Attention Mechanisms presented in Section 2.5 help us address such a sequential problem and how masking is used to enforce hard constraints on the decision rules the policy outputs. Figure 5.5 gives a high-level overview of MARDAM, and how the four blocks composing it interact. Every following subsection describes one of the blocks that need to encode part of the environment state.

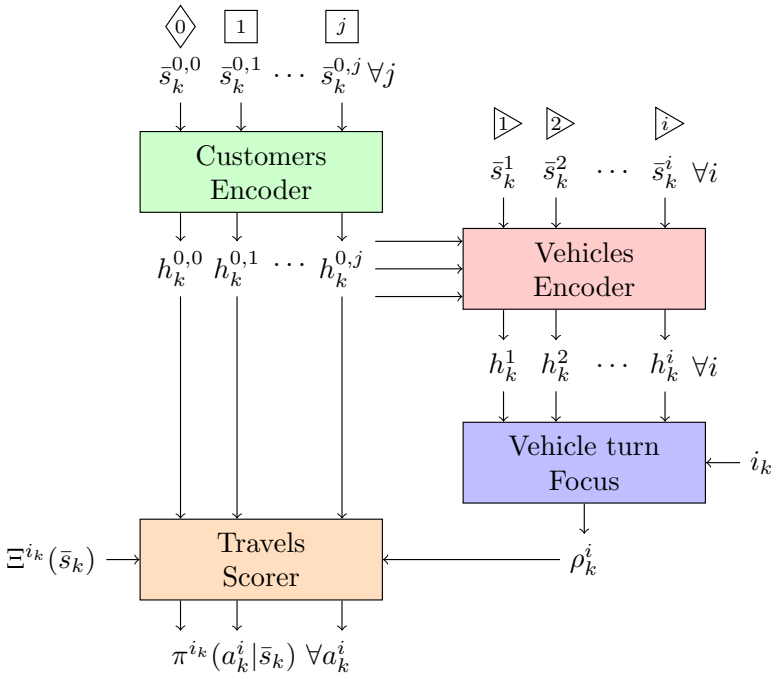


Figure 5.5 – Overview of MARDAM - Four blocks composes this policy architecture, each specializing in one of the factor of the state. At the end, it produces a decision rule giving the probabilities of choosing any pending customer as the next destination for the vehicle that has just finished its previous service.

The "customers encoder" block has the critical role to represent the customers features which is key to characterize the problem we are facing. It is implemented as a Transformer encoder [112] that can efficiently learn a representation of the fully connected graph formed by the customers as discussed in Section 2.5. It outputs an internal representation $h_k^{0,j}$ for the depot and every customer j. All other internal representations in MARDAM are based on this initial customer encoding. Indeed vehicles states and actions are closely related to the customer features.

The "vehicles encoder" block uses an Attention Mechanism to map each vehicle state $\bar{s}_k^i$ to its own internal representation h_k^i, built as a weighted combination of the customers encoding. Contrary to existing approaches based on Deep Neural Networks (DNN) found in the literature [50, 73, 31], we model the problem as a multi-agent decision process. Instead of encoding the partial route of a single vehicle, we base our policy directly on the factored state of all agents. It enables us to provide online decision rules to any agent in a dynamic environment, contrary to existing method which can only be used offline to plan multiple routes. The "agent turn focus" combines all internal representations of the vehicles states into a single internal state representation ρ_k^i for the vehicle i_k taking a decision at the current step k using an attention mechanism driven by the internal representation of the current agent.

Finally the "travels scorer" block matches each customer representation $h_k^{0,j}$ with the internal state representation of current vehicle ρ_k^i using a learned measure similarly to how Attention Mechanisms compute the weights to combine their input values. As the set of actions directly maps to the set of customers, we do not create a new encoding for them, but simply use the one of the customers coming from the first block. To take into account only valid action, the resulting scores are masked using the binary vector given by $\Xi^{i_k}(\bar{s}_k)$.

5.2.1 Customers encoding

The "customers encoder" block is the first entry point of our policy. This encoder has to learn to extract useful features from the shared state $\bar{s}^0$. In this case, it learns to represent the fully-connected graph formed by the depot 0 and the customers $j \in [\![1, n]\!]$. It identifies patterns relating their features to one another, and provides an enriched vectorial representation $h_k^{0,j}$ of every node j (depot or customer). This representation is driven by the experiences and rewards encountered during training. For example, some components of these learned representations could correspond to a notion of priority for customers with early and tight TWs, other could score them in term of relative demands, or classify them into spatial clusters which could help pre-allocating them to the vehicles. Nonetheless, because an encoder based on DNN is completely learned and is capable of approximating almost any kind of function, the representations we obtain do not necessarily have such a strict and interpretable semantic. They are mostly abstract vectors which might make sense only for the layers using then afterwards, even though they contain all the important information.

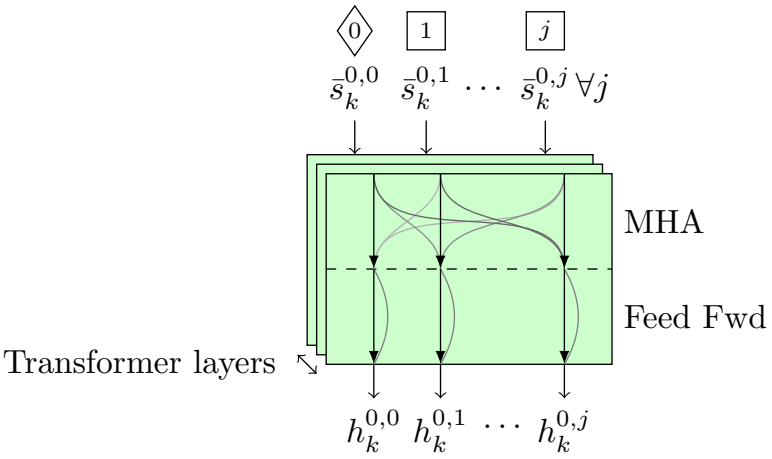


Figure 5.6 – Customer encoder block in MARDAM implemented as a Transformer encoder to create internal representation $h_k^{0,j}$ for a set of customers described by features vectors $\bar{s}_k^{0,j}$.

As we developed in Chapter 2, many kinds of DNN layers could have fulfilled the role of a graph encoder to represent the set of nodes (depot and customers) that constitute the shared state of our sMMDP formalization of DS-VRPs. Similarly to [50, 31], we chose to exploit the flexibility and efficiency of the Transformer encoder introduced in [112], illustrated in the context of MARDAM on Figure 5.6. The Transformer is flexible because the same set of weights can be used for inputs of different sizes, making it capable of generalizing to configurations of VRPs involving varying number of customers. Its efficiency comes from the fact that its operations are invariant to permutations of its input, hence it can naturally capture the nature of unordered sets without having to learn that all permutations of their elements are equivalent.

The Transformer encoder is an alternating stack of Multi-Head Attention (MHA) layers and fully connected (a.k.a. linear) layers, which are all short-circuited by skip connections and batch-normalized. We refer our readers to Figure 2.13 in Chapter 2 for more details on the operations involved in a MHA layer. We developed our own implementation of Transformer encoder before it was integrated in `pytorch`, one of the leading library for DNN we have chosen to use during this thesis. An efficient implementation of the MHA layer only declares one set of weights for all heads, taking care of initializing them per block, to perform all operations in parallel heads at once, then reshape the results. It takes advantage of the caching and parallelization of batch

operations which are well optimized in the standard low level linear algebra libraries for GPUs, such as `CUDA` and `CUBLAS`.

We picture the shared state of the customers as a way to provide a common context to the agents. Hence the encoding resulting from this first block will be carried along the other blocks to enrich their input with more information. In particular, it will complement the individual state of every agent to form individual intermediate representations relating to all factors an agent can influence through its actions.

5.2.2 Vehicles encoding

The "vehicles encoder" block in MARDAM also learns to extract features from the state of the system. The role of this block is to independently produce an intermediate representation in a high dimensional space for every agent $i \in \mathcal{I}$ at any point of the trajectory. This individual intermediate representation takes as input both the individual state of all the vehicles $\bar{s}^1, \bar{s}^2, \ldots, \bar{s}^i$ and the context coming from the shared state $\bar{s}^0$ through its encoding h^0.

Combined with the "agent turn focus" block which comes right after it, it constitutes the key difference with existing DNN approaches to routing problems [50, 73]. Together they enable us to model the VRP as a true multi-agent system, instead of reformulating it as a single vehicle problem. Adopting a multi-agent model is required to be able to adapt our decision rules online for all vehicles given the stochastic and dynamic information we gather while executing them.

Closest to our approach is the Attention Model (AM) of Kool, Hoof, and Welling [50], which has been previously discussed in Section 2.5. It creates a context vector directly from the customers embeddings produced by the Transformer. This context can only describe one route at a time and cannot represent the state of multiple vehicles evolving in parallel. This auto-regressive representation could be considered as a representation of the history of actions taken by the vehicle.

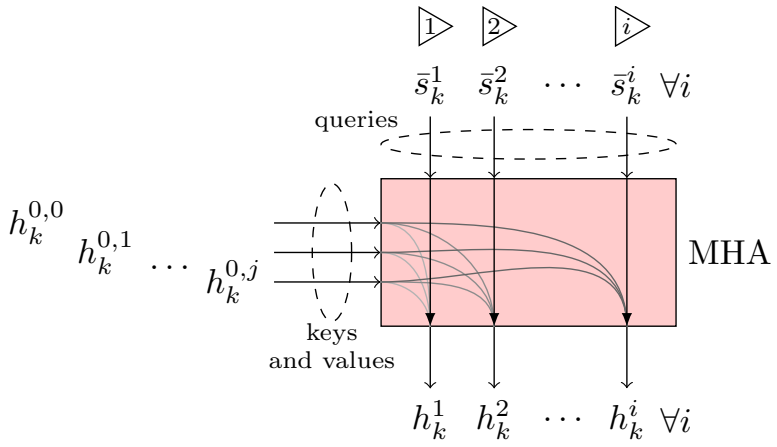


Figure 5.7 – Vehicles encoder block in MARDAM implemented as a MHA layer to create internal representation h_k^i for a set of vehicles based on their states $\bar{s}_k^i$ and the internal representations of the customers $h_k^{0,j}$.

To circumvent this limitation, we propose to learn an intermediate representation directly from the continuous states of all the vehicles in the fleet using a MHA layer, which was described in details in Section 2.5 of Chapter 2. Each individual state $\bar{s}_k^i$ is used as a query on the set

of internal representations $h_k^{0,j}$ of the customers, playing the role of both keys and values. In every head of the MHA layer, a learned measure scores queries against keys to output weighted combinations of the values. The results of the different heads are then mixed together through a fully connected layer. We obtain an internal representation h_k^i for all vehicles $i \in [\![1, m]\!]$, which regroups the information coming from the individual state of each agent and the context provided by the shared state, i.e. the features of the customers, which characterizes the problem.

This block can take advantage of potential symmetries between agents. In our case, we consider an homogeneous fleet of vehicles, hence the role of each vehicle could be swapped with another one without impacting the objective value, and all individual states $\bar{s}^i$ lie in the same space $\bar{\mathcal{S}}^i$. That is another advantage of the MHA layer that motivated our design choice. The parameters of the query projections in the MHA of the "vehicles encoder" block are shared between agents, which improve learning efficiency. It also means that MARDAM can be trained to solve problems with a fleet of varying size, which helps it generalize to problems with more agents.

The internal representations h_k^i of the vehicles are then focused on the vehicle i_k currently taking a decision, which is given by the environment through the turn function σ. The "agent turn focus" block also uses a MHA to combine the representations of every agent given the shared context, i.e. the global state of the system, into a single individual internal state vector ρ_k^i.

5.2.3 Turn focus

It is in the "Vehicle turn focus" block that we take into account the sequentiality of our model. It takes as input the set of intermediate representations produced above and the current turn value i_k to produce the individual internal representation of the state for the agent currently taking a decision. Figure 5.8 shows how the intermediate representation $h_k^{i_k}$ of the agent i_k currently acting is selected and compared to all the other intermediate representations, in order to form the internal state ρ_k^i of the agent. Similarly to the previous "vehicles encoder" block, this block is able to take a variable number of inputs in order to adapt to different number of agents.

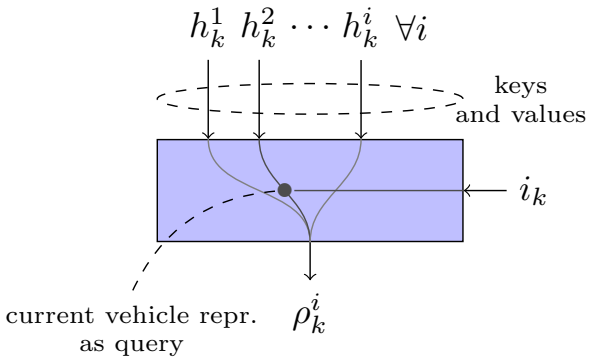


Figure 5.8 – Turn focus block of MARDAM. From the internal representation of every agents, it focuses on the one currently taking a decision given by the turn variable i_k. It uses a MHA layer to generate an internal state representation for this agent given the intermediate representation of the whole fleet.

This block is meant to help the coordination between agents by extracting the features that customize the decision rule for this specific agent, isolating it from the rest of them. It is implemented using a MHA layer. Intermediate representations $h_k^{i'}$ $\forall i'$ play the role of keys and values. The one of the agent i_k currently acting drives the attention as a query vector $h_k^{i_k}$. The query and the keys are projected to a different vector space in multiple parallel heads, where they can be scored against each other using a simple scalar product. These scores are normalized and used as weights to combine projections of the values associated to the keys. The projection of the query and the keys corresponds to learn a metric comparing them to one another. The multiple heads provide some variety and some redundancy that help the layer explore the class of function it can learn and extract more relevant features.

As shown on the bottom-left of Figure 5.5, we obtain a single vector ρ_k^i which is a combination of the intermediate states of all agents, driven by the one of the agent currently acting. ρ_k^i will play the role of the individual internal state for this agent when computing a decision rule in the next block.

5.2.4 Travels scorer

From the individual internal state ρ_k^i output by the previous block, the "travels scorer" block has to produce an individual decision rule $\pi^{i_k}(a_k^i|\bar{s}_k)$ indicating the probabilities of choosing any valid action $a_k^i \in \Xi^{i_k}(\bar{s}_k)$ for the current agent i_k given the current state $\bar{s}_k$. In the case of MARDAM, we can take advantage of the fact that the set of individual actions directly maps to the set of customers. Hence, we can use the internal customers representation $h_k^{0,j}$ as encoding of the individual actions to be matched against the individual internal state ρ_k^i.

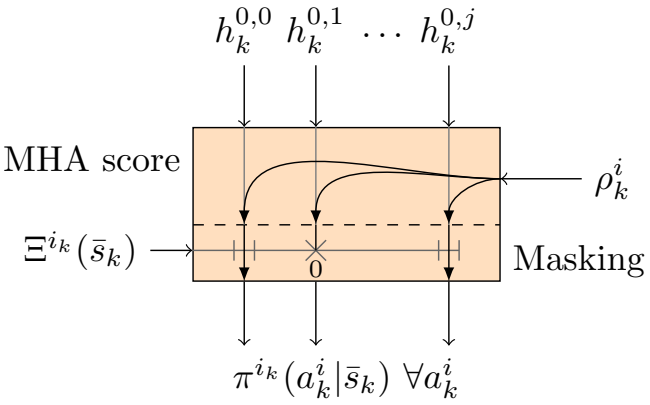


Figure 5.9 – Travels scorer block in MARDAM implemented as the scoring part of a head in a MHA layer with masking. It outputs an individual decision rule $\pi(\cdot|\bar{s}_k)$ for the agent i_k currently acting based on its internal state representation ρ_k^i and the internal representation $h_k^{0,j}$ of the customers.

To enforce that only valid actions can be selected, the resulting scores are masked such that customers that cannot be served by the current vehicle have a zero probability to be chosen as next target. Using masking, we avoid having to re-encode only the valid customers at every decision step. This greatly improves the efficiency of MARDAM, because customers representations are only updated when the set of customers varies (in case of dynamic appearance), and all projections depending on them in the vehicles encoder and travels scorer blocks are also pre-computed.

5.3 Conclusion

In this chapter, we used the sMMDP model we introduced in Chapter 4 to model a Dynamic and Stochastic VRP with Capacitated vehicles and Time Windows (DS-CVRPTW). We consider three main sources of uncertainty and dynamism: stochastic customers whose features are randomly sampled from a-priori unknown distributions; stochastic travel times which also follows an a-priori unknown distribution; and dynamic customers appearing while vehicles are travelling. We first extensively explained how we model this rich DS-VRP problem relying on tensor structures compatible with Deep RL frameworks and easily parallelized into batches to reduce the variance of the gradient estimation and improve training efficiency. We discussed relevant modelling choices including state structure and reward model.

Then we presented a novel DNN architecture based on Attention Mechanisms which we call MARDAM. It is designed as a modular policy decomposed into four blocks. Each of them is responsible for representing part of the state of the system. This architecture is capable of encoding both the shared state of the environment, which represents a distribution over different customers configurations on which we can generalize, and the dynamic state of the whole fleet of vehicles to adapt to uncertain events as customers appearance or delays while executing the route. The representation power offered by the Attention Mechanisms enabled us to learn policies that generalize to multiple scenarios and instances of DS-CVRPTW, and output online decision rules contrary to existing architectures which do not capture the multi-agent dimension of the problem.

Each block in MARDAM can evolve independently from the other blocks in order to adapt to new problem variants or inherit from other fields making progress on representing unordered set of vectors. From this initial architecture, it should be possible to deepen some of the blocks, especially the vehicles encoder and the turn focus, to enrich the class of functions they are capable of representing. Additionally, one of our short-term perspective is to explore how we could revisit MARDAM to handle problems where vehicles have to take their decisions in a decentralized way, only based on local partial observation of the state. In this case, we would also like to reconnect to our preliminary work on Centralized Training for Decentralized Control (CTDC) [2] which we presented in Chapter 3. This implies redesigning the vehicle encoder to only take as input local individual observations, and compress the individual history of the vehicle using recurrent layers. This also means we need to extract the centralized fleet representation such that the resulting policies stay separable, however in the CTDC paradigm, we could still benefit from this extra information on the whole state of the system during training.

In the next chapter, we evaluate our model and policy architecture. We report the results we obtained using MARDAM on a set of artificially generated benchmarks compared to more traditional heuristics used in the OR literature.

Chapter 6

Experimental Evaluation

After reviewing existing approaches to address DS-VRPs in Chapter 2, we developed in Chapter 4 a new specialization of Multi-agent Markov Decision Processes (MMDP) which we call sequential MMDP (sMMDP). In this model, agents act sequentially meaning that we can isolate the effect of their individual actions on the states, and separate the rewards they receive into individual contributions. We then developed a new policy network architecture called MARDAM in Chapter 5. This policy is designed around four blocks which specialize into representing their own factor of the state. It takes advantage of the multi-agent and sequential structure of the problem.

To test the performances and desirable properties of MARDAM, we implement artificial benchmarks on different variants of VRPs. We start by reviewing existing benchmark sources, which do not fit our needs for large sets of Dynamic and Stochastic scenarios in Section 6.1. Then we describe how we generate our benchmarks, and which methods we use to address them in Section 6.2. Once the setup is defined, we test the flexibility of MARDAM and its capacity to update its internal representations when customers appear dynamically along the routes in DS-CVRPTWs in Section 6.4. Secondly, we test the adaptability and robustness offered by MARDAM against stochastic travel times in S-CVRPTWs in Section 6.5. To further evaluate the generalization capabilities of MARDAM, and compare it to existing DL approaches [50, 73], we run an experiment on deterministic CVRPs and CVRPTWs in Section 6.6. We then study the impact of lateness penalty and how we can emulate hard TWs in Section 6.7. Afterwards, we show that MARDAM is sufficiently flexible to address problems of different sizes with the same set of weights in Section 6.8. We published all these results in IEEE Transactions on Intelligent Transportation Systems [1]. Finally we will open up the discussion about more realistic logistic simulation built upon micro-traffic simulation, which we initially presented in [5].

6.1 Useful web resources

There are a lot of resources online to obtain benchmarks for many variants of VRPs, because the OR literature is not used to open-source its algorithms, but instead often provides tables comparing approaches on pre-defined benchmarks. However, there are not many benchmarks available for rich DS-VRPs [48] especially not in the quantity required by Deep Learning approaches. In this section, we will review four sources of benchmarks freely available online, namely VRP-REP, Solomon benchmarks, TSPLIB and VRPLIB.

VRP-REP (http://www.vrp-rep.org) It might be the most useful resource available online, as it references many variants of problems, and proposes a standardized format for the instances definitions, and best known solutions. It also references for each dataset the articles that declared using the benchmarks. Among the variety of datasets, some are modelled from real-life scenarios while other are purely artificial.

Solomon benchmarks (http://web.cba.neu.edu/~msolomon/problems.htm) One of the first benchmark we considered. It was introduced in [102] and consists in artificial instances of CVRPTW with 100 customers. It considers three different patterns for their positions – uniformly random, clustered, or a mix of both– in an area of 100×100 units. For each of these three spatial patterns, different temporal scenarios are studied: horizons ranging from short (240 units) to long (3390 units), with various distributions of time windows (ratios of constrained customers, wide or narrow TW width, ...) Customers' demands ranges from 1 to 50 units, and are matched against three configurations of a homogeneous fleet of 25 vehicles, with a capacity of 200, 700 or 1000 units. All instances are feasible, and depending on the combination of features as presented above, the optimal routes will be mostly constrained by capacities or time windows, or any intermediate balance between both.

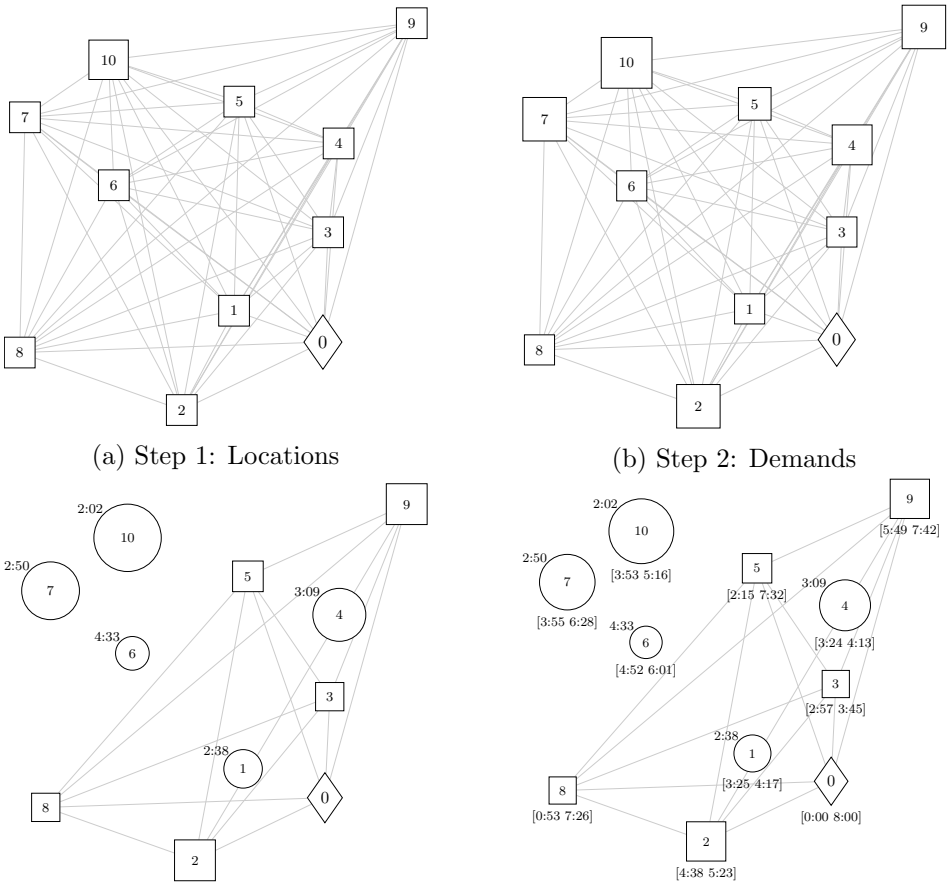
TSPLIB and VRPLIB (http://elib.zib.de/pub/mp-testdata) These two datasets propose a unified and standardized format for a few variants of static routing problems, initially designed to be solved by mathematical programming, specifically Mixed Integer Linear Programming (MILP). Most instances are modelled from real life logistic scenarios.

None of the datasets mentioned above met our need for dynamic and stochastic environments, in a large amount of configurations, and representing various situations. Indeed, DNNs require a large amount of data to train, and it would be counter-productive to overfit on a few hundred samples, as can be found in standard benchmark, when we want to test our capability to generalize to new problems. That is why we developed our own DS-CVRPTW artificial instances generator and an RL dynamic environment from which we could sample trajectories and train our policy.

6.2 Experimental setup

To train MARDAM, with the objective of generalizing to any instance of DS-CVRPTWs, we need a rich dataset containing thousands of scenarios with different customers positions, demands, time window distributions, and appearance patterns. The data generation procedure decomposes into 4 steps, as illustrated in Figure 6.1.

1. First, we sample the locations of the depot and customers from a discrete uniform distribution in an area of $100 \times 100\text{km}^2$. This spatial distribution of customers' location is quite straightforward, and was used previously in most existing work applying DNN to routing problems [50, 6, 73].
2. Next step associates to each customer a discrete demand sampled uniformly between 0.5 and 4m^3 with 1-decimal precision. We fix the capacity of our vehicles to 20m^3. If necessary, we scale down all demands such that the total capacity of the fleet can satisfy the total demand of all customers.



(a) Step 1: Locations

(b) Step 2: Demands

(c) Step 3: Dynamic appearance

(d) Step 4: Time windows

Figure 6.1 – Illustration of data generation for DS-CVRPTWs with 10 customers. The diamond node represents the depot, labelled 0. All other nodes are customers, labelled from 1 to 10, whose sizes map to their demand. Square nodes are static customers known before the vehicles leave the depot. Circle nodes are dynamic customers appearing after vehicles deployment, for which the upper (x:xx) label indicates appearance time. The lower label ([x:xx y:yy]) indicates the time windows.

3. Then, we sample the appearance times of dynamic customers from a distribution characterized by two parameters, similarly to [8]: a) the *degree of dynamism* p_{dyn}; b) the fraction of customers that appear early. For a horizon of 8h, early customers will appear in the first 2h40, late customers between 2h40 and 5h20, and we suppose no more orders are accepted afterwards. This early/late ratio parameter gives us some control on the length of the partial routes that have already been committed to when new customers appear.

4. Finally, we sample the time windows limiting when customer can be served, as proposed in the classical CVRPTW benchmark of [102]. A ratio controls how many customers are constrained. We start by sampling a service duration between 5 and 30min and a time window width between 30min and 1h30. Then we randomly position the window such that the ready and due times stay coherent with the earliest the customer can be reached at, and the latest the service can start to return to depot in time. We can adjust the ratio of customer constrained by time windows, and sampling their width first makes it possible to control how tight these windows are, and constraint the trajectories in a way that makes stochastic travel times critical.

We implemented an artificial environment from which we can sample trajectories based on these generated data and our sMMDP model of DS-CVRPTWs. MARDAM is trained using this environment on a training dataset of more than a million customer configurations. All model and training meta-parameters are summarized in Table 6.1.

Model params	Customers features size	D_C	$\in \{3, 6, 7\}$
	Vehicles states size	D_V	$= 4$
	Inner model dimension	D	$= 128$
	Head dimension	D_H	$= 16$
	Number of heads	N_H	$= 8$
	Feed-forward hidden dim.	D_F	$= 512$
	Number of encoder layers	N_L	$= 3$
	tanh saturation	C_{tanh}	$= 10$
Training meta-params	Number of training samples	B_{train}	$= 1280000$
	Number of testing samples	B_{test}	$= 128$
	Number of validation samples	B_{valid}	$= 10000$
	Mini-batch size	B	$= 512$
	Number of iteration / epoch	N_{iter}	$= 2500$
	Number of training epoch	N_{epoch}	$= 100$
	Actor learning rate	α^{π}	$= 0.0001$
	Critic learning rate	α^{bl}	$= 0.001$
	Max gradient norm	$\lvert\nabla\rvert_{\max}$	$= 2$
	Optimizer		Adam[49]

Table 6.1 – Model and training meta-parameters used in all experiments.

```
generate dataset ;
initialize policy ;
foreach training epoch do
    foreach mini-batch in dataset do
        foreach instance in mini-batch do /* ran in parallel */
            sample trajectory using policy ;
            estimate values of encountered states using critic ;
        estimate mean Policy Gradient on mini-batch of trajectories ;
        estimate MSE of critic on mini-batch of trajectories ;
        back-propagate ;
        update policy and critic parameters ;
```

Algorithm 6.1: Actor-Critic algorithm to train MARDAM on DS-CVRPTW

6.3 Typical learning curves

Before presenting the results in terms of performances for the different benchmarks we developed, we will discuss how the architecture behaved during training in our sMMDP environment. We train MARDAM using the Actor-Critic algorithm detailled in Algorithm 6.1 where we alternate between an evaluation and an update phase. We sample a batch of trajectories using the current policy to estimate its value (critic) and the gradient of the objective w.r.t. its parameters (actor).

A typical run of such an algorithm produces the learning curves depicted in Figure 6.2. Each epoch on the horizontal axis corresponds to a pass through the entire training dataset. It took

approximately 6 minutes with our hardware setup[1]. The expected cumulated reward estimated by the critic matches the observed rewards on the training set right from the first epoch, with a slight tendency to overestimate the policy's performances. It is not surprising, as it is only an evaluation of the current policy, focused on a subset of states encountered during exploration. The Actor-Critic loss (AC loss) and gradient norm are related and slowly tend towards 0 as the policy converges.

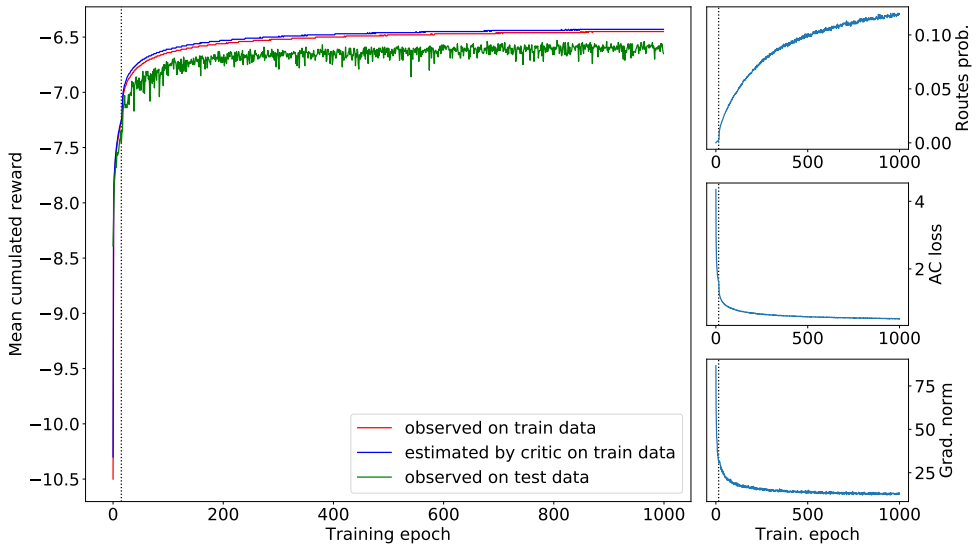


Figure 6.2 – Learning curves of MARDAM on a static and deterministic CVRPTW with 20 customers.

On all our curves, we notice a mode switch after approximately 15 epochs, especially on the mean cumulated rewards and the mean routes/trajectories probabilities. On the former, we can identify two phases: before epoch 15, we have a very fast improvement phase and after that, we switch to an exponential stabilization phase. On the latter, the first 15 epochs are exploratory, and no particular sequences of actions are reinforced. After 15 epochs, we enter an exponential growth phase. We think that this switch appears when the policy has learned that leaving customers pending at the end of a trajectory is highly penalized, causing a discontinuity in the reward signal. For the first 15 epochs, vehicles return to depot early and the reward is dominated by a term proportional to the number of customers left pending. Afterwards the policy can focus on improving the routes in term of travelled distances and lateness.

Once trained, we test it on a test dataset of ten thousand problems, sampled independently of the training set. We have 2 decoding schemes to exploit the decision rules output by MARDAM. "MARDAM (g)" greedily chooses the action with maximum probability at each step. "MARDAM (s)" samples actions from the decision rules to obtain 100 different trajectories, then keeps the best one.

We obtain most of our baselines using Google's ORTools [1] backed by their CP-SAT solver. It outputs plans for each instance of the validation sets which are then executed in our simulated environment. "ORTools (d)" dynamically replans the parts of the routes that have not been committed to every time a new customer appears. "BestIns" is a greedy online insertion heuristic we implemented. It uses ORTools to initialize the routes with the customers known a-priori. Then it tries to insert any new customer where it minimizes the detour cost and discards it if this cost goes over a fixed threshold. "ORTools (o)" uses an optimistic constant speed V_{nom} to

[1]It took 3min/epoch for $N = 10$, and 22min/epoch for $N = 50$.

compute travel times, while "ORTools (e)" uses the expected value $E[V_t] = (1 - p_{\text{slow}})V_{\text{nom}} + p_{\text{slow}}V_{\text{slow}}$ of this speed, of the speed distribution illustrated on Figure 5.4.

We include 2 state-of-the-art approaches based on DL as baselines for deterministic CVRP: the Attention Model (AM) of [50] and the Recurrent Neural Network (RNN) of [73]. Both architectures are trained using the code provided by the authors. For evaluation, "AM (g)" and "AM (s)" follows the same decoding schemes as MARDAM. "RNN (g)" is also greedy, while "RNN (bs)" uses a beam search of width 10. For the sake of comparison, we also report the results of the Lin-Kernighan-Helsgaun [45] heuristic, noted "LKH3".

All results are reported as the mean and standard deviation of the total cumulated rewards on the problem instances of the test dataset. Cumulated rewards include travelled distances plus lateness and pending customers penalties. For DS-CVRPTW, we also report separately the average Quality of Service (QoS) i.e. the fraction of customer served. Otherwise, percentages indicate relative gaps compared to the starred methods in each table row. Bold fonts highlight best values.

6.4 DS-CVRPTW

In this first experiment, we test the flexibility of MARDAM and its capacity to update its internal representations when customers appear dynamically along the routes in DS-CVRPTWs. We want to evaluate how MARDAM balances QoS and costs compared to 2 simple online baselines. Our results for increasing degrees of dynamism p_{dyn} are reported in Table 6.2.

Table 6.2 – MARDAM compared to ORTools and BestIns in terms of travel and lateness cost, and QoS on DS-CVRPTWs instances of various sizes and degrees of dynamism

p_{dyn}	dim. / Method	$N = 10$ (Cost)	$N = 10$ (QoS)	$N = 20$ (Cost)	$N = 20$ (QoS)	$N = 50$ (Cost)	$N = 50$ (QoS)
0%	ORTools (d)	3.97 ± 0.61	**100.00%**	$\mathbf{6.28 \pm 0.79}$	**100.00%**	$\mathbf{12.11 \pm 1.36}$	**100.00%**
	MARDAM (g)	$\mathbf{3.89 \pm 0.58}$	99.94%	6.35 ± 0.77	99.95%	12.77 ± 1.34	99.95%
$\leqslant 40\%$	BestIns	4.79 ± 1.53	94.69%	12.59 ± 5.73	98.05%	-	-
	ORTools (d)	5.87 ± 1.64	96.20%	10.93 ± 2.75	98.64%	26.85 ± 7.08	**99.71%**
	MARDAM (g)	$\mathbf{4.63 \pm 0.86}$	**99.81%**	$\mathbf{7.37 \pm 1.01}$	**99.92%**	$\mathbf{17.24 \pm 3.26}$	99.30%
$< 60\%$	BestIns	$\mathbf{4.79 \pm 1.59}$	89.38%	15.55 ± 8.37	96.17%	-	-
	ORTools (d)	5.83 ± 1.59	94.96%	11.21 ± 2.77	98.17%	27.54 ± 7.42	**99.61%**
	MARDAM (g)	4.93 ± 0.97	**99.70%**	$\mathbf{7.70 \pm 1.07}$	**99.89%**	$\mathbf{17.89 \pm 3.69}$	98.38%
$\geqslant 60\%$	BestIns	$\mathbf{4.40 \pm 2.05}$	79.86%	17.97 ± 9.42	92.28%	-	-
	ORTools (d)	5.87 ± 1.57	**94.53%**	11.20 ± 2.64	97.94%	28.41 ± 7.95	**99.54%**
	MARDAM (g)	4.69 ± 2.09	86.78%	$\mathbf{8.17 \pm 1.17}$	**99.85%**	$\mathbf{18.64 \pm 4.13}$	91.54%

"BestIns" was too slow to properly run on larger instances. Its cost / QoS balance is driven by the threshold on the insertion cost. As no existing part of the routes get questioned, the insertion cost can quickly exceed the threshold. That's why it has a relatively poor QoS for a relatively low cost avoiding detours on smaller problems. However its performance quickly degrades with the problem dimension.

Although "ORTools (d)" could use disjunctions to ignore customers, it seems it emphasized the QoS over travel and lateness costs of detours. MARDAM maintains relatively low costs while keeping a competitive QoS for low and intermediate degree of dynamism, with a slight degradation for $p_{\text{dyn}} \geqslant 60\%$. It naturally balances QoS and costs because of the reward model we used during training. Hence MARDAM is capable of efficiently mitigating costs degradation without sacrificing too much on the QoS, especially as the dimension of the instances increases.

6.5 S-CVRPTW

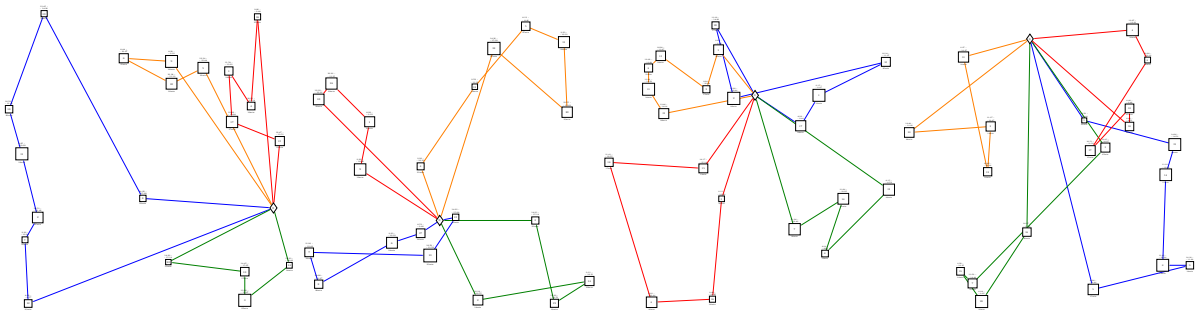
Next, we test the adaptability and robustness offered by MARDAM against stochastic travel times in S-CVRPTWs. As an online policy, we expect it to mitigate delays and behave better than the offline planning baselines "ORTools (o)" and "ORTools (e)" which can only use a-priori statistics. Table 6.3 compares MARDAM and ORTools for increasing probabilities of slow down p_{slow}.

We can see that MARDAM takes the best over both baselines when the slow down probability and the entropy of the vehicle speed increase. Even if "ORTools (e)" seems more robust than "ORTools (o)", it still cannot adapt to given realizations of the random variables and tends to accumulate lateness that propagates along routes on tight schedules. This mostly explains the relative advantage of an online policy such as MARDAM compared to pre-planned routes. However, we note that MARDAM seems to have difficulties scaling up with the number of customers, as its performances are not as favorable for larger instances ($N = 50$) and low slow probability ($p_{\text{slow}} < 30$). We think that this indicates some limitations in the capacity of MARDAM to represent larger instances, as we can notice the same discrepancies on deterministic CVRPs in the final experiment of Section 6.6.

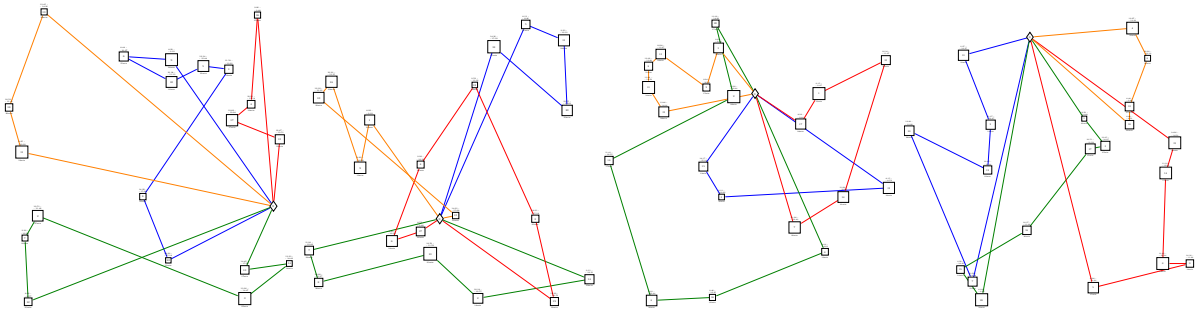
Table 6.3 – MARDAM compared to ORTools in terms of travel, lateness and pending cost on S-CVRPTWs instances of various sizes and probabilities of slowdown

p_{slow}	Method \ dim.	$N = 10$	$N = 20$	$N = 50$
0%	*ORTools (o)	4.39 ± 0.70	6.98 ± 0.90	$\mathbf{13.38 \pm 1.55}$
	MARDAM (g)	$\mathbf{4.24 \pm 0.64}(-\mathbf{3\%})$	$\mathbf{6.91 \pm 0.84}(-\mathbf{1\%})$	$13.66 \pm 1.45(2\%)$
5%	*ORTools (o)	4.47 ± 0.72	7.11 ± 0.91	13.64 ± 1.59
	ORTools (e)	$4.47 \pm 0.72(-0\%)$	$7.07 \pm 0.91(-0\%)$	$\mathbf{13.58 \pm 1.58}(-\mathbf{0\%})$
	MARDAM (g)	$\mathbf{4.31 \pm 0.65}(-\mathbf{4\%})$	$\mathbf{7.00 \pm 0.83}(-\mathbf{1\%})$	$13.84 \pm 1.44(2\%)$
10%	*ORTools (o)	4.55 ± 0.75	7.22 ± 0.95	13.87 ± 1.67
	ORTools (e)	$4.55 \pm 0.74(-0\%)$	$7.20 \pm 0.95(-0\%)$	$\mathbf{13.84 \pm 1.68}(-\mathbf{0\%})$
	MARDAM (g)	$\mathbf{4.37 \pm 0.67}(-\mathbf{4\%})$	$\mathbf{7.11 \pm 0.86}(-\mathbf{1\%})$	$14.02 \pm 1.49(1\%)$
20%	*ORTools (o)	4.72 ± 0.81	7.47 ± 1.04	14.39 ± 1.84
	ORTools (e)	$4.72 \pm 0.81(0\%)$	$7.46 \pm 1.03(-0\%)$	$\mathbf{14.37 \pm 1.85}(-\mathbf{0\%})$
	MARDAM (g)	$\mathbf{4.52 \pm 0.72}(-\mathbf{4\%})$	$\mathbf{7.33 \pm 0.92}(-\mathbf{2\%})$	$14.41 \pm 1.59(0\%)$
30%	*ORTools (o)	4.90 ± 0.88	7.74 ± 1.12	14.94 ± 2.01
	ORTools (e)	$4.90 \pm 0.87(0\%)$	$7.72 \pm 1.12(-0\%)$	$14.92 \pm 2.04(-1\%)$
	MARDAM (g)	$\mathbf{4.67 \pm 0.76}(-\mathbf{5\%})$	$\mathbf{7.58 \pm 1.00}(-\mathbf{2\%})$	$\mathbf{14.86 \pm 1.72}(-\mathbf{1\%})$
50%	*ORTools (o)	5.31 ± 1.04	8.37 ± 1.33	16.25 ± 2.44
	ORTools (e)	$5.27 \pm 0.99(-1\%)$	$8.28 \pm 1.29(-1\%)$	$16.07 \pm 2.41(-1\%)$
	MARDAM (g)	$\mathbf{5.01 \pm 0.87}(-\mathbf{5\%})$	$\mathbf{8.14 \pm 1.17}(-\mathbf{3\%})$	$\mathbf{15.92 \pm 2.05}(-\mathbf{2\%})$

6.6 CVRP and CVRPTW



(a) MARDAM



(b) ORTools

Figure 6.3 – Comparison between routes sampled using MARDAM and routes obtained by ORTools on a few instances of CVRPTW. Each column corresponds to the same customer configuration for MARDAM and ORTools. If parts of some routes seem almost identical, ORTools solutions sometimes look more convoluted than the ones of MARDAM. This might be caused by the former being stricter on lateness than the latter, even though both consider soft TWs.

In this experiment, we are looking for a reliable point of comparison where classical approaches and heuristics are known to give near optimal results. Figure 6.3 provides a few illustrations of routes obtained by MARDAM compared to routes returned by ORTools on a random selection of a few CVRPTW instances. Both methods respect capacity constraint while assigning customers to vehicles. Even though TWs are considered soft in both cases, MARDAM looks more forgiving on lateness and emphasizes shorter distances, while ORTools seems stricter on the customers due time, hence causing more detours to limit lateness. That being said, we want to compare the generalization capability of MARDAM to other state-of-the-art DL approaches on a larger set of validation instances. As we can see in Table 6.4, MARDAM is keeping up with offline planning methods on the CVRP, but scales poorly with the number of customers and vehicles. We believe it is not an intrinsic limitation of the method but more a question of engineering and fine-tuning hyper-parameters.

Surprisingly, while competitive on the largest instances, the performances of RNN quickly degrade the smaller the problem is. We suppose that this behavior is due to the absence of hard limitation on the number of vehicles involved. If this does not make a high difference to add an extra vehicle when the total demand of 50 customers already requires 10, it is much more detrimental to start one extra route when no more than 2 vehicles are required to cover the demand of 10 customers. Even if not constraining the number of vehicles either, AM does not have the same difficulties, and displays better performances than MARDAM except when $N = 10$.

Table 6.4 – MARDAM compared to LKH3, ORTools, AM and RNN in term of travel, lateness and pending cost on CVRP and CVRPTW instances of various sizes

Pb.	Method \ dim.	$N = 10$	$N = 20$	$N = 50$
CVRP	*LKH3	3.40 ± 0.44	5.39 ± 0.56	10.64 ± 1.16
	ORTools (o)	$3.41 \pm 0.51(0\%)$	$5.41 \pm 0.60(0\%)$	$10.67 \pm 1.19(0\%)$
	AM (g)	$3.47 \pm 0.50(2\%)$	$5.39 \pm 0.59(-0\%)$	$10.42 \pm 1.12(-2\%)$
	AM (s)	$3.39 \pm 0.48(-0\%)$	$\mathbf{5.28 \pm 0.57(-2\%)}$	$\mathbf{10.15 \pm 1.09(-5\%)}$
	RNN (g)	$4.74 \pm 0.80(41\%)$	$6.50 \pm 0.82(22\%)$	$11.21 \pm 1.26(6\%)$
	RNN (bs)	$4.60 \pm 0.77(37\%)$	$6.39 \pm 0.83(19\%)$	$10.98 \pm 1.24(4\%)$
	MARDAM (g)	$3.44 \pm 0.49(1\%)$	$5.78 \pm 0.70(7\%)$	$11.24 \pm 1.27(6\%)$
	MARDAM (s)	$\mathbf{3.38 \pm 0.47(-1\%)}$	$5.46 \pm 0.57(1\%)$	$10.88 \pm 1.20(2\%)$
CVRPTW	*LKH3	4.05 ± 0.67	6.17 ± 0.81	$\mathbf{11.40 \pm 1.35}$
	ORTools (o)	$3.98 \pm 0.59(-5\%)$	$6.27 \pm 0.76(-1\%)$	$12.09 \pm 1.38(5\%)$
	MARDAM (g)	$3.87 \pm 0.56(-7\%)$	$6.33 \pm 0.75(0\%)$	$12.68 \pm 1.32(10\%)$
	MARDAM (s)	$\mathbf{3.82 \pm 0.54(-9\%)}$	$\mathbf{6.06 \pm 0.67(-4\%)}$	$12.05 \pm 1.23(4\%)$

When adding TW constraints, MARDAM presents favorable results for smaller instances, while getting much closer for the largest ones. We believe that the penalty cost used in LKH3 to take into account TW constraints during its search might not be well adapted to soft TW, and the instances we generated are not guaranteed to be solvable with hard TW.

6.7 Lateness penalties and hard TW constraints

We preferred considering soft time windows because of stochastic travel times, and because it was easier to model as a reward signal. Nonetheless, as CVRPTW is traditionally solved with hard time windows, we wanted to investigate how MARDAM would learn to penalize lateness more and more, until a prohibitive cost would enforce service strictly in time. We had to also increase the cost of pending customers to avoid situations where the policy would prefer completely ignore a customer instead of risking to serve it late.

In the following experiment, as reported in Table 6.5, we show how a policy trained with low lateness penalties (MARDAM soft) and one trained with high lateness penalties (MARDAM hard) behave compared to the Lin-Kernighan-Helsgaun heuristic, which preferably outputs solutions which do not violates any constraint (unless the local search finds none, our generated instances are not guaranteed solvable with hard TWs). As expected, MARDAM (soft) compromises a lot on lateness to reduce distances, hence it performs poorly in environment with high penalties. MARDAM (hard) is almost as good as LKH3 to meet TWs, at the cost of slightly less optimal distances. Further fine-tuning balancing the cost of lateness compared to travelled distances might enable MARDAM to reach global performances as good as LKH3.

Table 6.5 – Mean and standard deviation of the cost for soft or hard lateness penalty

Method	Total Distance	Total Lateness	Total Cost (low penalty)	Total Cost (high penalty)
LKH3	6.25 ± 0.92	$\mathbf{0.02 \pm 0.05}$	$\mathbf{6.27 \pm 0.94}$	$\mathbf{6.47 \pm 1.25}$
MARDAM (soft)	$\mathbf{6.06 \pm 0.70}$	0.27 ± 0.24	$6.34 \pm 0.78(2\%)$	$8.79 \pm 2.58(36\%)$
MARDAM (hard)	6.67 ± 0.85	0.03 ± 0.04	$6.69 \pm 0.87(7\%)$	$6.93 \pm 1.08(8\%)$

We also implemented a small variant of the algorithm where the lateness penalty was considered as a varying parameter along the different training epochs. It starts small in order to help exploration and slowly increase until it reaches its prohibitive value. When experimenting with this variant, we did not observe noticeable differences in performance in term of mean cost or convergence time compared to learning directly with the highest late penalty.

6.8 Avoiding dimension specialization

In all our previous experiments, MARDAM was trained separately on every problem dimension, such that we had specific sets of weights for $N = 10$, $N = 20$ and $N = 50$. Because of this specialization, we expect any of our trained policy to not generalize properly to problems of different dimensions.

However a solution to overcome this limitation is not hard to find: we simply need to train MARDAM on a dataset containing problems of any size. Ideally we would like to make mini-batches with various number of customers, to avoid biasing our gradients estimates, which is not straightforward because the tensors we use to describe a mini-batch of customers have fixed dimensions. Luckily, we already have a solution for this small technical difficulties: customer masking. Using the same mechanism we implemented to hide dynamic customers, we can permanently mask dummy padding customers, which are filled with zeros to match the mini-batch dimension, so that their contributions to all intermediate activations in the network is null.

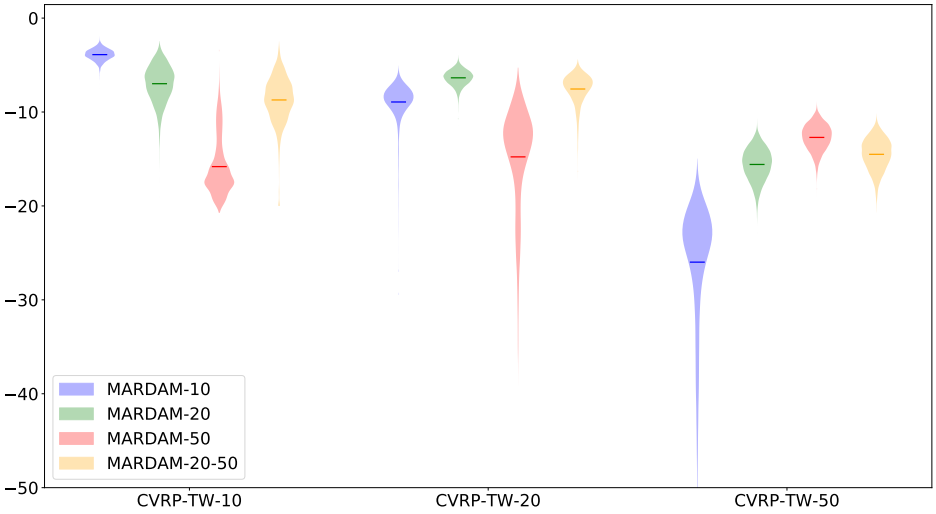


Figure 6.4 – Distribution of total cumulated rewards on 3 different problem sizes (10, 20, 50 horizontally) for MARDAM trained using only instances with 10, 20, or 50 customers, and for MARDAM trained with a dataset containing instances with 20 to 50 customers. This variety in the training dataset helps generalizing better across dimensions.

Figure 6.4 illustrates our result using this varied dataset generation and masking procedures compared to our previous trainings. As we can see, each network trained on a specific dimension behaves at its best only when solving problems matching the dimension its trained on, and its performances quickly degrade when the number of customers differs. On the contrary the network trained with instances of various sizes maintains pretty good performances all across the

board. Moreover, we would like to highlight that this dimension-generalizing network did not benefit from more training epochs: all 4 networks were trained with the same hyper-parameters, only the range of numbers of customers during dataset generation differs. More interestingly, we notice that even though the MARDAM-20-50 network did not see instances with only 10 customers during training, its degradation in performance is still limited compared to the other training conditions. We suppose that introducing variety in the training dataset widen the range of dimension the network can generalize to even further than the range available during training. We also expect this variety in training samples to also help the network address dynamic problems, because the subset of customers considered at each time step is also not of fixed size.

6.9 Towards more realistic scenarios

Training our policy on artificial data is only a first step to address real-world routing problems online. Indeed, our artificial benchmark with euclidean travel costs and independent random travel times does not capture the dynamics of a dense city center. To address this issue while still preserving the ability to generate and sample a large training dataset, we introduce in this section an alternative benchmark based on the micro-traffic simulator SUMO [52]. The benchmark relies on a framework we developed and called SULFR (pronounced "sulfur") for Simulation of Urban Logistics For Reinforcement learning. We will start by describing how SULFR works, then report some preliminary results obtained after training MARDAM with it.

6.9.1 The SULFR framework

We designed SULFR as an interface between the simulator SUMO and a learning algorithm such as Actor-Critic. It was originally developed for Pickup and Delivery Problems (PDP) a variant of VRPs where vehicles do not resupply at a common depot but must first pickup a package at a node before delivering it to another one. We also included support for electric vehicles, which have to plan for recharge at pre-defined charging stations. We later focussed on the setup we worked on with MARDAM and adapted SULFR to DS-CVRPTWs with the objective to minimize total travelled distance and lateness penalties.

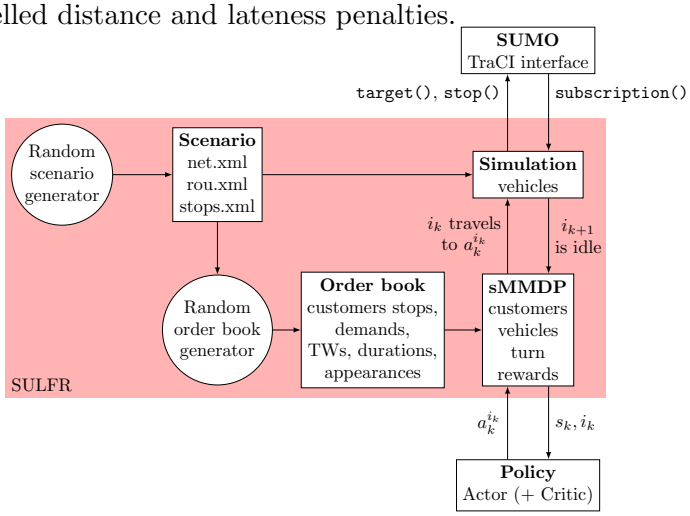


Figure 6.5 – High level overview of SULFR, our interface between a learning algorithm such as Actor-Critic and the SUMO micro-traffic simulator.

Its components are described in Figure 6.5. We provide tools to generate random scenarios and customers configurations (referred to as order books) but the users can also directly provide their own training dataset based on historical data. From these data, we spawn a batch of SUMO processes that will simulate the operations of our fleet of delivery vehicles in different road network layouts, and dynamic traffic conditions. The states of all simulations are monitored and controlled in parallel by a sMMDP environment model through the TCP interface of SUMO called TraCI.

Thanks to SUMO, we simulate the operations of a fleet of delivery vehicles in different configurations. We can vary the city or district in which the vehicles evolve which are described as SUMO `net.xml` files. They are associated with a set of potential customers addresses listed as SUMO container stops in `add.xml` files. We only use these stops to locate the customers on the road networks and do not rely on the simulation of logistics integrated in SUMO to handle customer services, to keep more control over vehicles assignment and routing. Last elements that are required to setup the SUMO simulations are descriptions of the dynamic traffic conditions. They consist in lists of uncontrolled vehicles that start travelling from one location to another at a given time. The traffic flow is then an emerging property of their interactions on the road when each one of them obeys simple car-following and lane-switching rules [53].

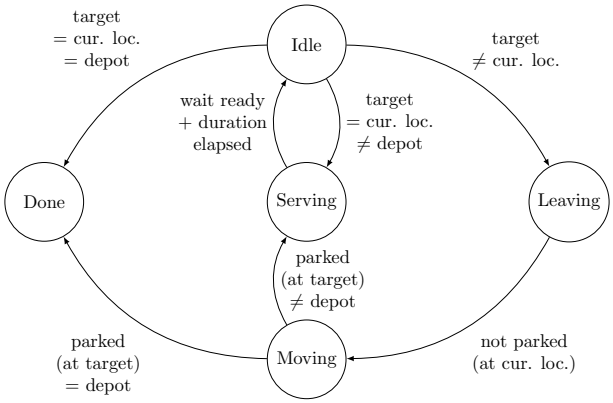


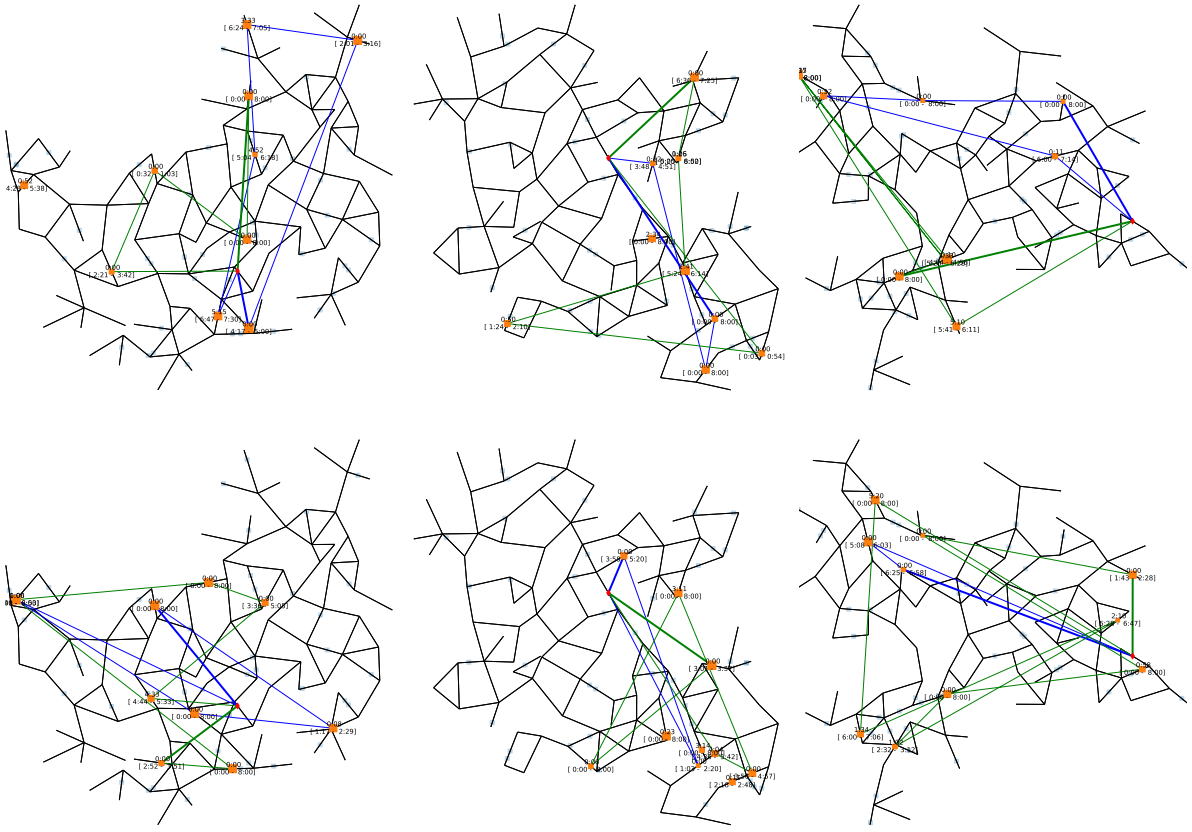
Figure 6.6 – Finite state machine governing a vehicle when it travels from one customer to the next, based on the information available from the TraCI interface of SUMO.

We rely on the router included in SUMO to make the agent travel from one customer location to the next. The customer services are simulated by SULFR while the vehicles simply wait at their stops in SUMO. Vehicles are handled by the finite state machine depicted in Figure 6.6. The upper-level policy we are training gets queried for the next action (i.e. a target, either the next customer to serve or returning to the depot) every time the vehicle is in the "idle" state, and all other transition are triggered by feedbacks from the TraCI connection provided by SUMO.

If the user does not provide any specific scenario, the random scenario generator creates a random graph following a power-law for nodes connectivity to be used as a road network, selects random addresses for the customers and makes random trips from pairs of edges at starting at a varying frequency. The random order book generator creates a set of customers features very similarly to our previous benchmarks. The only difference is that the locations are now sampled from the set of potential customers addresses of the scenario. The demands, time windows, durations and appearances are still sampled as indicated in Section 6.2.

6.9.2 Preliminary results training MARDAM with SULFR

We trained MARDAM for 5000 iterations using batches of size 4 on 16 different scenarios. It represents a very small number of updates to our policy parameters, but even if ran in parallel, the simulation remains quite slow. This 5000 iterations already took a few days of training, mostly waiting for the simulator to run. We did not reach convergence in that time, and would need at least 50 times more iterations to get the same number of parameters updates as in our artificial benchmarks. However, as a proof of concept, we report in Figure 6.7 a selection of routes sampled using this early training of MARDAM on 3 random scenarios of our pool.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 6.7 – Selection of routes obtained by MARDAM after 5000 iterations on 3 different scenarios, and 2 different random order books per scenario. The depot is represented as a red diamond, and all active customers are orange squares. The routes followed by vehicles 1 and 2 are indicated as blue and green lines, respectively.

We show that MARDAM was capable of learning a policy which can adapt to any of the three scenarios, and represent different configuration of customers described in randomly sampled order books. We are still far from optimal routes in all cases, and we would need many more update iterations or better sample efficiency to reach proper results.

6.10 Conclusion

In this chapter, we developed benchmarks where we could train and evaluate MARDAM, our specialized policy network based on Attention Mechanisms, which we previously presented in Chapter 5. The representation power offered by the Attention Mechanisms enabled us to learn policies that generalize to multiple scenarios and instances of DS-CVRPTWs, and output online decision rules competitive with both classical solvers such as ORTools and LKH3 and recent DNN-based approaches, namely AM [50] and RNN [73]. Our MARDAM architecture is fast, robust and flexible compared to myopic a-priori routes. It can efficiently adapt to real-time perturbation and appearing customers.

This generalization power comes with a requirement for rich training datasets. We are not aware of any DS-CVRPTW dataset large enough to properly train a DNN architecture, and realistic problem instances of this kind are currently lacking [48]. To tend towards this goal, we developed a new environment called SULFR interfacing with the micro-traffic simulation SUMO [52]. There are still a lot of open questions on the proper way to represent the road networks and the customers address, but Graph Encoders [4] combined with Attention Mechanisms look like promising solutions. Another point which still needs to be addressed is the low sample efficiency of Deep RL, because as it stands, the simulator is too slow to sample many training trajectories and obtain proper results in reasonable time.

However, this original data cost is compensated by the adaptation and generalization capabilities of the MARDAM policy. On top of this, we wish to extend our training procedure and improve our architecture to enable MARDAM to keep learning online, while being deployed on a real fleet of vehicles. If this creates a risk of over-fitting to the last scenarios encountered, it can also make MARDAM keep improving and specializing on the situations it has to face the most.

On a final thought, MARDAM is currently built as a policy where all agents have full observability. To improve its robustness in more realistic scenarios where perceptions and communication might be imperfect or delayed, we want to extend our model and policy to support partial local observability and individual internal representation for every agents.

Chapter 7

Conclusion

The goal of this thesis was to explore how we could address rich Vehicle Routing Problems (VRPs) in a stochastic and dynamic environment using the modern and promising tools offered by Deep Reinforcement Learning. The first step was to established the state of the art on Dynamic and Stochastic VRPs (DS-VRPs). We started by reviewing the exact methods such as the modern Branch-and-Cut-and-Price of [79], which have found provably optimal solutions to larger and larger instances of the static and deterministic VRP. However these exact methods do not adapt very well to the dynamic and stochastic case, which multiplies the number of decision variables, required to describe solutions for every potential outcome and dynamic evolution of the system. Complementary to these exact approaches, the OR literature is rich of many efficient heuristics which cannot guarantee an optimality gap but can very efficiently explore the space of solutions for larger problems, with a variety of operational constraints. There application to DS-VRPs is still subject to some limitations, and often consists in solving new frozen problems from scratch every time their solutions are invalidated by new dynamic and stochastic events, in framework such as the Multiple Scenario Approach [8].

After this state of the art review, we decided to explore an alternative approach based on Reinforcement Learning [105], which naturally describes dynamic and stochastic problems. The challenge was to find a way to represent the state of the system. Indeed, we wanted to train policies that could adapt to various customers configurations, and represent the state of our fleet of vehicles. To this end, we drew inspiration from a recent line of works initiated by [116] who used Deep Neural Network to learn to solve routing problems directly from data. The architectures kept improving to more efficiently represent fully-connected graphs of customers for which current state of the art is to use a Transformer encoder [50, 31]. Nonetheless, all existing approaches based on Deep RL have reframed the VRP as a single-vehicle problem, and do not take into account its multi-agent dimension. Hence, they cannot represent the global state of the system in a dynamic and stochastic environment, where multiple vehicles evolve in parallel.

7.1 Contributions

Actor-Critic for Decentralized Control. We first started by investigating Policy Gradient methods for multi-agent systems in the general case of Decentralized Partially-Observable Markov Decision Processes (Dec-POMDPs). We focussed on a particular training paradigm which we call Centralized Training for Decentralized Control (CTDC). In this paradigm, we consider two separate phases for policy training and policy deployment. During training, we have access to additional information, similarly to how a football coach will observe and give instructions to his players during training, so that they can learn strategies which they will execute during a match without his interventions. We developed the mathematical foundations of Actor Critic algorithm in the CTDC paradigm, extending the Policy Gradient Theorem [106] and the Critic compatibility conditions to this setup. We also highlighted the relation to Natural Actor Critic algorithms. We validated our approach using traditional benchmarks of the Dec-POMDP literature, and showed a net advantage to the CTDC paradigm compared to fully centralized or fully decentralized approaches. This work was also the occasion to start discussing the problem of individual internal state representation. We compared two different history compression methods: truncated history and Recurrent Neural Network (RNN), and noticed better performances on the latter in term of objective value and convergence speed. The individual internal state approximation problem is still an open issue, but RNN showed promising capabilities.

Sequential MMDP. After this preliminary work on multi-agent policy gradient methods in the general Dec-POMDPs case, we developed a novel multi-agent model which could describe the rich DS-VRPs we want to address. We introduced a new variant of Multi-agent MDP model which we call sequential MMDP (sMMDP). It captures the property of routing problems that all agents do not choose their decision simultaneously, but instead target a new customer only when they are done serving the previous one. It could be applied to any other planning problem where agents actions describe tasks with a temporal dimension, whose effects on the state of the system can be isolated. We derived variants of the Bellman equation for sMMDP and analyzed its complexity compared to its equivalent simultaneous counterpart (MMDP). Our experiments tend to show a net advantage to using a sMMDP for problems which naturally have a sequential structure. However the sequentiality constraint we impose on the model slightly reduces its expressivity compared to the general MMDP, hence it is only adapted to such cases.

MARDAM. In the last part of the thesis, we proposed to apply our sMMDP model to DS-VRPs. As the state and action spaces become much larger than what we considered previously, we designed a new DNN architecture to implement a parametric policy. This policy we called MARDAM decomposes into four blocks, as shown on Figure 7.1. Each of them corresponds to one of the factor of the state in our sequential model of DS-VRPs, namely the set of customers features, the set of vehicles states, the current vehicle turn, and the subset of valid individual actions. We extensively use Multi-Head Attention layers and the Transformer encoder of [112]. The former helps us deal with sets of customers and vehicles of various size, with the interesting property to be invariant to permutation of its input which improves its sample efficiency compared to recurrent structures. The latter can capture the structure of the customers configurations which result from a priori unknown random distributions, and extract patterns from the graph they form.

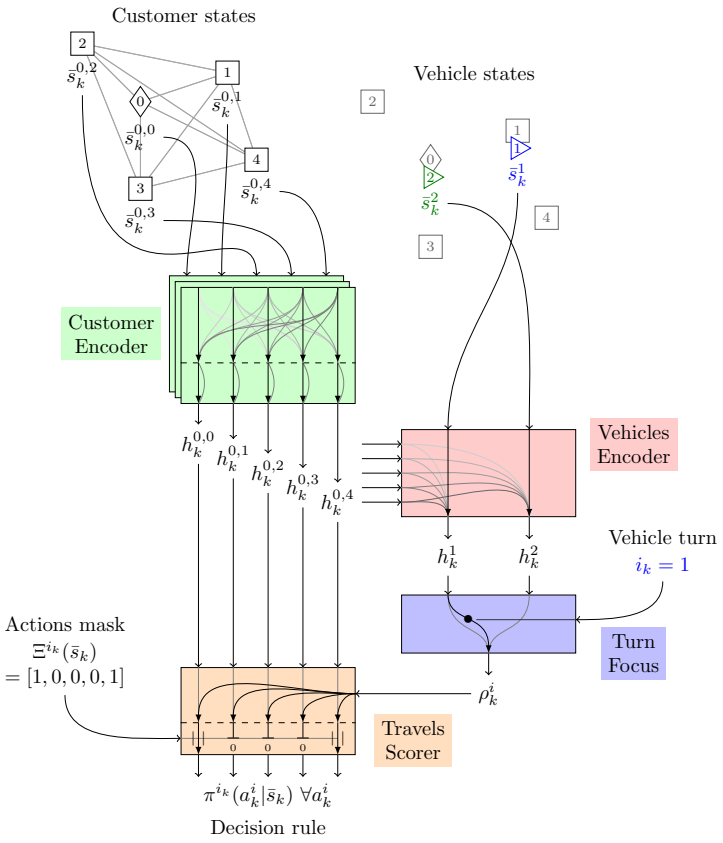


Figure 7.1 – Summary of the MARDAM architecture which learns to represent the multi-agent state of a DS-VRP with varying sets of customers and vehicles, to provide online decision rules and choose individual actions in a sequential decision process.

To the best of our knowledge, we were the first to take into account the multi-agent dimension of VRPs with a Deep RL approach. We reported promising experimental results in term of adaptability, robustness, and objective value on a set of artificially generated benchmarks inspired by existing generation processes [102, 8]. We demonstrated the overall quality of our approach, in particular for Dynamic and Stochastic VRP compared to ORTools, a solver developed by Google AI which uses Adaptive Large Neighborhood Search backed by their CP-SAT solver. We also developed a more involving benchmark based on the micro-traffic simulation SUMO [52], a software which Volvo also experimented with to evaluate the energy consumption of their vehicles while travelling on existing city road networks. We reported preliminary results as a proof of concept training MARDAM on a small selection of scenarios corresponding to different road networks and customers set.

7.2 Perspectives

There are still a lot of open questions about the application of Deep RL techniques to real-world routing problems. We have identified multiple axes of improvement for future research.

Combining Deep RL and other heuristics. First, Deep RL is really data-hungry, and all commonly used training algorithms are not very sample-efficient. In order to avoid relying on artificially generated dataset to get enough samples to train DNNs, it is important to improve the efficiency of Deep RL algorithms while making sure to avoid over-fitting. For example, instead of Stochastic Gradient Descent, recent approaches [103, 104] have successfully trained DNNs using a simple evolutionary algorithm, which drastically reduced the number of training iterations required to reach similar performances.

There are other opportunities to combine Deep RL and traditional heuristics from the OR literature. One could train a DNN to produce initial solutions before applying local search operator to improve it, as was proposed in [31]. However synergies between Deep RL and OR heuristics are not limited to such a two-phase approach. For example, being able to quickly sample batches of solutions in a single forward pass of a DNN could prove very useful and efficient in any framework relying on sets of solutions, including but not limited to lookup search, Genetic Algorithms, Ant Colony Optimization, and Multiple Scenario Approach. The challenge in such hybrid approaches is to redesign the reward signal to be able to keep applying Deep RL algorithms or find new ways to update the parameters of the network with nice convergence properties.

Model scalability and policy refinement. The next axis for improvement which appeared clearly in our final experiments is to work on the scalability of our approach. Even though MARDAM is built around Multi-Head Attention layers that use the same set of parameters for inputs of varying sizes, its ability to address larger problem seems limited. There is certainly a problem of representation capabilities for graphs of customers and fleets of vehicles of larger sizes. This could be solved by deepening the architecture, but also comes with technical difficulties. If sharing parameters and exploiting the nature and structure of the states in the blocks composing MARDAM help reducing the number of parameters, we are limited in the number of intermediate activations we can store by the memory available on our GPUs. Deepening the network would require compromises on the batch size, increasing the variance of the gradient estimations. One way to address this issue is to take into account the synergies between a global level of decision, for example from the point of view of the dispatcher managing the whole fleet, and a local level of decision for each vehicle. In a way, such hierarchical levels of decision have already proven useful in [100, 16], who used state aggregation to enable their Approximate Dynamic Programming approach to scale up to thousands of customers. There is already a few tools in the RL literature that can help us model this hierarchy and global-local interaction, such as Options [107, 3], Feudal Learning [29, 114], or MaxQ [34].

Additionally, we have not fully exploited the Multi-Agent Policy Gradient theory we developed in the Centralized Training for Decentralized Control (CTDC) framework [2] of Section 3.3. Indeed, our current sMMDP model for DS-VRPs is fully observable, hence it did not require a decentralized control. However, vehicles might only have partial and local observability in real-world scenarios, in which case it is necessary to consider more complex model, such as Dec-POSMDP [76], the semi-Markov variant of Dec-POMDP, which takes into account durative actions. The partial observability and decentralized control constraints also impose a refinement

of the policy architecture, which would need to be separable, and may benefit from a distinct centralized critic structure, to capture the additional information available at training in the CTDC framework, as proposed in [37] for example.

Realistic DS-VRPs and state representation. The last perspective we want to discuss here is the continuation of the work we initiated with our SULFR simulation framework. First, we think there is more engineering to do to improve the interface and the parallel management of the simulations, using frameworks such as Asynchronous Advantage Actor Critic (A3C) [70] to avoid waiting for all simulations at each step. On a side note, there are also potential improvements on some of the mechanisms integrated in SUMO, namely the integrated router and the side-parking. Otherwise SUMO is a very promising tool, which can simulate very rich scenarios with road infrastructures, traffic lights, public transport lines, trains, and pedestrians.

More importantly, we are going to keep searching for efficient and compact state representations for realistic DS-VRPs using DNNs. To go further than customers in an euclidean space, and take into account the underlying road infrastructures, we think it is necessary to add an encoder level that learn to represent these networks of streets and intersections. A Transformer would not be the best solution for this task, as it is much more efficient for fully-connected graphs, and more general Graph encoder architecture [4] would be more adapted. Once we obtain an internal representation for the city or district where the vehicles will evolve in, Attention Mechanisms seem like a good candidate to learn representations for the customers and vehicles locations.

List of Publications

Journals

[1] Guillaume Bono, Jilles Steeve Dibangoye, Olivier Simonin, Laëtitia Matignon, and Florian Pereyron. "Solving Multi-agent Routing Problems using Deep Attention Mechanisms". In: *IEEE Transactions on Intelligent Transportation Systems* (2020), pp. 1–10.

Conferences

[2] Guillaume Bono, Jilles Steeve Dibangoye, Laëtitia Matignon, Florian Pereyron, and Olivier Simonin. "Cooperative Multi-agent Policy Gradient". In: *Machine Learning and Knowledge Discovery in Databases - Proceedings of the European Conference on Machine Learning*. Ed. by Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim. 2018, pp. 459–476.

[3] Guillaume Bono, Jilles Steeve Dibangoye, Laëtitia Matignon, Florian Pereyron, and Olivier Simonin. "Classification des problèmes stochastiques et dynamiques de collectes et de livraisons par des véhicules intelligents". In: *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes*. 2017.

[4] Guillaume Bono, Jilles Steeve Dibangoye, Laëtitia Matignon, Florian Pereyron, and Olivier Simonin. "Sur le Gradient de la Politique pour les Systèmes Multi-agents Coopératifs". In: *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes*. 2018.

Workshops

[5] Guillaume Bono, Jilles Steeve Dibangoye, Laëtitia Matignon, Florian Pereyron, and Olivier Simonin. "Simulation of Stochastic Vehicle Routing Problems in Urban Environment". In: *Prediction and Generative Modeling in Reinforcement Learning*. Ed. by Matteo Pirotta, Roberto Calandra, Sergey Levine, Martin Riedmiller, and Alessandro Lazaric. 2018.

Reports

[6] Guillaume Bono, Jilles Steeve Dibangoye, Laëtitia Matignon, Florian Pereyron, and Olivier Simonin. *On the Study of Cooperative Multi-Agent Policy Gradient*. Tech. rep. INSA Lyon, INRIA, 2018. URL: https://hal.inria.fr/hal-01821677.

Bibliography

[1] Google AI. *Google's Operations Research Tools*. online. 2018. URL: https://developers.google.com/optimization/.

[2] Christopher Amato, Jilles Steeve Dibangoye, and Shlomo Zilberstein. "Incremental Policy Generation for Finite-Horizon Dec-POMDPs". In: *Proceedings of the 19th International Conference on International Conference on Automated Planning and Scheduling*. Ed. by Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis. 2009, pp. 2–9.

[3] Pierre-Luc Bacon, Jean Harb, and Doina Precup. "The Option-Critic Architecture". In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. Ed. by Satinder Singh and Shaul Markovitch. 2017, pp. 1726–1734.

[4] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. *Relational inductive biases, deep learning, and graph networks*. arXiv. 2018. eprint: 1806.01261 (cs.LG).

[5] Richard Bellman. "Dynamic Programming". In: *Science* 153.3731 (1966), pp. 34–37.

[6] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. *Neural Combinatorial Optimization with Reinforcement Learning*. arXiv. 2016. eprint: 1611.09940 (cs.AI).

[7] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. *Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon*. arXiv. 2018. eprint: 1811.06128 (cs.LG).

[8] Russell W. Bent and Pascal Van Hentenryck. "Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers". In: *Operations Research* 52.6 (2004), pp. 977–987.

[15] Craig Boutilier. "Planning, Learning and Coordination in Multiagent Decision Processes". In: *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*. Ed. by Yoav Shoham. 1996, pp. 195–210.

[16] Belgacem Bouzaiene-Ayari, Clark Cheng, Sourav Das, Ricardo Fiorillo, and Warren B. Powell. "From Single Commodity to Multiattribute Models for Locomotive Optimization: A Comparison of Optimal Integer Programming and Approximate Dynamic Programming". In: *Transportation Science* 50.2 (2016), pp. 366–389.

[17] Steven J Bradtke and Michael O Duff. "Reinforcement learning methods for continuous-time Markov decision problems". In: *Advances in Neural Information Processing Systems*. Ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. 1995, pp. 393–400.

[18] Mu-Chen Chen, Yu-Hsiang Hsiao, Reddivari Himadeep Reddy, and Manoj Kumar Tiwari. "The Self-Learning Particle Swarm Optimization approach for routing pickup and delivery of multiple products with material handling in multiple cross-docks". In: *Transportation Research Part E: Logistics and Transportation Review* 91.1 (2016), pp. 208–226.

[19] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Tech. rep. Carnegie Mellon University, 1976.

[20] B. Coltin and M. Veloso. "Online pickup and delivery planning with transfers for mobile robots". In: *IEEE International Conference on Robotics and Automation.* Ed. by Ning Xi, William R. Hamel, and Jindong Tan. 2014, pp. 5786–5791.

[21] European Commission. *Region and Cities Illustrated.* online. 2018. URL: https://ec.europa.eu/eurostat/cache/RCI.

[22] Ryan G Conrad and Miguel Andres Figliozzi. "The Recharging Vehicle Routing Problem". In: *Proceedings of the 2011 Industrial Engineering Research Conference.* Ed. by T. Doolen and E. Van Aken. 2011.

[23] Jean-François Cordeau. "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem". In: *Operations Research* 54.3 (2006), pp. 573–586.

[24] Jean-François Cordeau, Guy Desaulniers, Jacques Desrosiers, Marius M. Solomon, and François Soumis. "VRP with Time Windows". In: *The Vehicle Routing Problem.* Ed. by Paolo Toth and Daniele Vigo. Society for Industrial and Applied Mathematics, 2002. Chap. 7, pp. 157–193.

[25] Jean-François Cordeau and Mirko Maischberger. "A parallel iterated tabu search heuristic for vehicle routing problems". In: *Computers and Operations Research* 39.9 (2012), pp. 2033–2050.

[26] Cristián E. Cortés, Doris Sáez, Alfredo Núñez, and Diego Muñoz-Carpintero. "Hybrid Adaptive Predictive Control for a Dynamic Pickup and Delivery Problem". In: *Transportation Science* 43.1 (2009), pp. 27–42.

[27] G. A. Croes. "A Method for Solving Traveling Salesman Problems". In: *Operations Research* 6.6 (1958), pp. 791–908.

[28] G. B. Dantzig and J. H. Ramser. "The Truck Dispatching Problem". In: *Management Science* 6.1 (1959), pp. 80–91.

[29] Peter Dayan and Geoffrey E Hinton. "Feudal Reinforcement Learning". In: *Advances in Neural Information Processing Systems.* Ed. by J. D. Cowan, G. Tesauro, and J. Alspector. 1993, pp. 271–278.

[30] Guy Desaulniers, Jacques Desrosiers, Andreas Erdmann, Marius M. Solomon, and François Soumis. "VRP with Pickup and Delivery". In: *The Vehicle Routing Problem.* Ed. by Paolo Toth and Daniele Vigo. Society for Industrial and Applied Mathematics, 2002. Chap. 9, pp. 225–242.

[31] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. "Learning Heuristics for the TSP by Policy Gradient". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research.* Ed. by Willem-Jan van Hoeve. 2018, pp. 170–181.

[32] Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. "Optimally Solving Dec-POMDPs as Continuous-State MDPs". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence.* Ed. by Sebastian Thrun, Francesca Rossi, Carla Gomes, and Umberto Grandi. 2013, pp. 90–96.

[33] Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. "Optimally Solving Dec-POMDPs as Continuous-State MDPs". In: *Journal of Artificial Intelligence Research* 55.1 (2016), pp. 443–497.

[34] Thomas G Dietterich. "Hierarchical reinforcement learning with the MAXQ value function decomposition". In: *Journal of Artificial Intelligence Research* 13.1 (2000), pp. 227–303.

[35] Moshe Dror, Gilbert Laporte, and Francois V. Louveaux. "Vehicle routing with stochastic demands and restricted failures". In: *Zeitschrift für Operations Research* 37.1 (1993), pp. 273–283.

[36] Sevgi Erdoğan and Elise Miller-Hooks. "A Green Vehicle Routing Problem". In: *Transportation Research Part E: Logistics and Transportation Review* 48.1 (2012), pp. 100–114.

[37] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. "Counterfactual Multi-Agent Policy Gradients". In: *Proceedings of the 32nd National Conference on Artifical Intelligence*. Ed. by Shlomo Zilberstein, Sheila McIlraith, and Kilian Weinberger. 2018, pp. 2974–2982.

[38] Michel Gendreau, Gilbert Laporte, and Jean-Yves Potvin. "Metaheuristics for the Capacitated VRP". In: *The Vehicle Routing Problem*. Ed. by Paolo Toth and Daniele Vigo. Society for Industrial and Applied Mathematics, 2002. Chap. 6, pp. 129–154.

[39] Fred Glover. "Tabu Search - Part I". In: *ORSA Journal on Computing* 1.3 (1989), pp. 190–206.

[40] Claudia V Goldman and Shlomo Zilberstein. "Decentralized control of cooperative systems: Categorization and complexity analysis". In: *Journal of Artificial Intelligence Research* 22.1 (2004), pp. 143–174.

[41] Ralph E. Gomory. "Outline of an algorithm for integer solutions to linear programs". In: *Bulletin of the American Mathematical Society* 64.1 (1958), pp. 275–278.

[42] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. "Cooperative Multi-agent Control Using Deep Reinforcement Learning". In: *Autonomous Agents and Multiagent Systems*. Ed. by Gita Sukthankar and Juan A. Rodriguez-Aguilar. 2017, pp. 66–83.

[43] Walter J. Gutjahr, Stefan Katzensteiner, and Peter Reiter. "A VNS Algorithm for Noisy Problems and Its Application to Project Portfolio Analysis". In: *Stochastic Algorithms: Foundations and Applications: 4th International Symposium*. Ed. by Juraj Hromkovič, Richard Královič, Marc Nunkesser, and Peter Widmayer. 2007, pp. 93–104.

[44] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. "Dynamic Programming for Partially Observable Stochastic Games". In: *Proceedings of the 19th National Conference on Artifical Intelligence*. Ed. by George Ferguson and Deborah McGuinness. 2004, pp. 709–715.

[45] Keld Helsgaun. *An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems*. Tech. rep. Roskilde University, 2017.

[46] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[47] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. "Learning Combinatorial Optimization Algorithms over Graphs". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. 2017, pp. 6348–6358.

[48] G. Kim, Y. Ong, C. K. Heng, P. S. Tan, and N. A. Zhang. "City Vehicle Routing Problem (City VRP): A Review". In: *IEEE Transactions on Intelligent Transportation Systems* 16.4 (2015), pp. 1654–1666.

[49] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. arXiv. 2014. eprint: 1412.6980 (cs.LG).

[50] Wouter Kool, Herke van Hoof, and Max Welling. "Attention, Learn to Solve Routing Problems!" In: *International Conference on Learning Representations*. Ed. by Tara Sainath, Alexander Rush, Sergey Levine, Karen Livescu, and Shakir Mohamed. 2019.

[51] Landon Kraemer and Bikramjit Banerjee. "Multi-agent reinforcement learning as a rehearsal for decentralized planning". In: *NeuroComputing* 190.1 (2016), pp. 82–94.

[52] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. "Recent Development and Applications of SUMO - Simulation of Urban MObility". In: *International Journal On Advances in Systems and Measurements* 5.3 (2012), pp. 128–138.

[53] S. Krauß. *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics*. Tech. rep. Institute of Transport Research - Hauptabteilung Mobilität und Systemtechnik, 1998.

[54] A. H. Land and A. G. Doig. "An Automatic Method of Solving Discrete Programming Problems". In: *Econometrica* 28.3 (1960), pp. 497–520.

[55] Gilbert Laporte, François Louveaux, and Hélène Mercure. "The Vehicle Routing Problem with Stochastic Travel Times". In: *Transportation Science* 26.3 (1992), pp. 161–170.

[56] Gilbert Laporte and François V. Louveaux. "The integer L-shaped method for stochastic integer programs with complete recourse". In: *Operations Research Letters* 13.3 (1993), pp. 133–142.

[57] John D. C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel. "An Algorithm for the Traveling Salesman Problem". In: *Operations Research* 11.6 (1963), pp. 972–989.

[58] Michael L. Littman and Csaba Szepesvári. "A Generalized Reinforcement-Learning Model: Convergence and Applications". In: *Proceedings of the 13th International Conference on Machine Learning*. Ed. by Lorenza Saitta. 1996, pp. 310–318.

[59] Michael Lederman Littman. "Algorithms for sequential decision making". PhD thesis. Brown University, 1996.

[60] Miao Liu, Christopher Amato, Emily P Anesta, John Daniel Griffith, and Jonathan P How. "Learning for Decentralized Control of Multiagent Systems in Large, Partially-Observable Stochastic Environments". In: *Proceedings of the 30th National Conference on Artifical Intelligence*. Ed. by Dale Schuurmans and Michael Wellman. 2016, pp. 2523–2529.

[61] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. 2017, pp. 6379–6390.

[62] Jacek Mańdziuk and Cezary Nejman. "UCT-Based Approach to Capacitated Vehicle Routing Problem". In: *Artificial Intelligence and Soft Computing: 14th International Conference*. Ed. by Leszek Rutkowski, Marcin Korytkowski, Rafal Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada. 2015, pp. 679–690.

[63] Peter Marbach, Oliver Mihatsch, Miriam Schulte, and John N Tsitsiklis. "Reinforcement learning for call admission control and routing in integrated service networks". In: *Advances in Neural Information Processing Systems*. Ed. by M. J. Kearns, S. A. Solla, and D. A. Cohn. 1998, pp. 922–928.

[64] Yannis Marinakis and Magdalene Marinaki. "A hybrid genetic - Particle Swarm Optimization Algorithm for the vehicle routing problem". In: *Expert Systems with Applications* 37.2 (2010), pp. 1446–1455.

[65] M. Mavrovouniotis and S. Yang. "Ant colony optimization with memory-based immigrants for the dynamic vehicle routing problem". In: *IEEE Congress on Evolutionary Computation*. Ed. by Hussein Abbass, Daryl Essam, and Ruhul Sarker. 2012, pp. 1–8.

[66] Stephan Meisel, Uli Suppa, and Dirk Mattfeld. "Serving Multiple Urban Areas with Stochastic Customer Requests". In: *Dynamics in Logistics: 2nd International Conference*. Ed. by Hans-Jörg Kreowski, Bernd Scholz-Reiter, and Klaus-Dieter Thoben. 2011, pp. 59–68.

[67] Martijn Mes, Matthieu van der Heijden, and Peter Schuur. "Interaction between intelligent agent strategies for real-time transportation planning". In: *Central European Journal of Operations Research* 21.2 (2013), pp. 337–358.

[68] Melanie Mitchell. *An Introduction to Genetic Algorithm*. Ed. by Bradford Books. MIT Press, 1998.

[69] N. Mladenović and P. Hansen. "Variable Neighborhood Search". In: *Computers and Operations Research* 24.11 (1997), pp. 1097–1100.

[70] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of the 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. 2016, pp. 1928–1937.

[71] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-level control through deep reinforcement learning". In: *Nature* 518.1 (2015), pp. 529–533.

[72] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati. "Ant Colony System for a Dynamic Vehicle Routing Problem". In: *Journal of Combinatorial Optimization* 10.4 (2005), pp. 327–343.

[73] MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. "Reinforcement Learning for Solving the Vehicle Routing Problem". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. 2018, pp. 9861–9871.

[74] Clara Novoa and Robert Storer. "An approximate dynamic programming approach for the vehicle routing problem with stochastic demands". In: *European Journal of Operational Research* 196.2 (2009), pp. 509–515.

[75] Frans A Oliehoek, Matthijs T J Spaan, Christopher Amato, and Shimon Whiteson. "Incremental Clustering and Expansion for Faster Optimal Planning in Dec-POMDPs". In: *Journal of Artificial Intelligence Research* 46.1 (2013), pp. 449–509.

[76] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P How, and John Vian. "Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions". In: *The International Journal of Robotics Research* 36.2 (2017), pp. 231–258.

[77] M. Padberg and G. Rinaldi. "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems". In: *SIAM Review* 33.1 (1991), pp. 60–100.

[78] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Ed. by Christos H Papadimitriou and Kenneth Steiglitz. Dover Publications, Inc., 1998.

[79] Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. "Improved branch-cut-and-price for capacitated vehicle routing". In: *Mathematical Programming Computation* 9.1 (2017), pp. 61–100.

[80] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Pack Kaelbling. "Learning to Cooperate via Policy Search". In: *16th Conference on Uncertainty in Artificial Intelligence*. Ed. by Craig Boutilier and Moisés Goldszmidt. 2000, pp. 1–8.

[81] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. "A review of dynamic vehicle routing problems". In: *European Journal of Operational Research* 225.1 (2013), pp. 1–11.

[82] Martin Pincus. "A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems". In: *Operations Research* 18.6 (1970), pp. 967–1235.

[83] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Ed. by Warren B Powell. John Wiley and Sons, 2007.

[84] Harilaos N. Psaraftis. "A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem". In: *Transportation Science* 14.2 (1980), pp. 130–154.

[85] Harilaos N. Psaraftis, Min Wen, and Christos A. Kontovas. "Dynamic vehicle routing problems: Three decades and counting". In: *Networks* 67.1 (2016), pp. 3–31.

[86] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. *Language Models are Unsupervised Multitask Learners*. OpenAI blog. 2019. URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[87] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *Conference on Computer Vision and Pattern Recognition*. Ed. by IEEE. 2017, pp. 6517–6525.

[88] Gerhard Reinelt. *TSPLIB*. online. 1995. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[89] Arpad Rimmel, Fabien Teytaud, and Tristan Cazenave. "Optimization of the Nested Monte-Carlo Algorithm on the Traveling Salesman Problem with Time Windows". In: *Evo**. Ed. by Jennifer Willies. 2011.

[90] Ulrike Ritzinger, Jakob Puchinger, and Richard F. Hartl. "A survey on dynamic and stochastic vehicle routing problems". In: *International Journal of Production Research* 54.1 (2016), pp. 215–231.

[91] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Ed. by Francesca Rossi, Peter Van Beek, and Toby Walsh. Elsevier, 2006.

[92] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[93] Doris Sáez, Cristián E. Cortés, and Alfredo Núñez. "Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering". In: *Computers and Operations Research* 35.11 (2008).

[94] Michael Saint-Guillain, Yves Deville, and Christine Solnon. "A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW". In: *Integration of AI and OR Techniques in Constraint Programming*. Ed. by Laurent Michel. 2015.

[95] M. Schilde, K.F. Doerner, and R.F. Hartl. "Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports". In: *Computers and Operations Research* 38.12 (2011), pp. 1719–1730.

[96] Michael Schneider, Andreas Stenger, and Dominik Goeke. "The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations". In: *Transportation Science* 48.4 (2014), pp. 500–520.

[97] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. 2015, pp. 1889–1897.

[98] Nicola Secomandi and Francois Margot. "Reoptimization Approaches for the Vehicle-Routing Problem with Stochastic Demands". In: *Operations Research* 57.1 (2009), pp. 214–230.

[99] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Ed. by Yoav Shoham and Kevin Leyton-Brown. Cambridge University Press, 2008.

[100] Hugo P. Simão, Jeff Day, Abraham P. George, Ted Gifford, John Nienow, and Warren B. Powell. "An Approximate Dynamic Programming Algorithm for Large-Scale Fleet Management: A Case Application". In: *Transportation Science* 43.2 (2009), pp. 178–197.

[101] Christine Solnon. *Ant Colony Optimization and Constraint Programming*. Ed. by John Wiley and Inc. Sons. Wiley Online Library, 2010.

[102] Marius M. Solomon. "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints". In: *Operations Research* 35.2 (1987), pp. 254–265.

[103] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. "Designing neural networks through neuroevolution". In: *Nature Machine Intelligence* 1.1 (2019), pp. 24–35.

[104] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. *Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning*. arXiv. 2017. eprint: 1712.06567 (cs.NE).

[105] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Ed. by Bradford Books. MIT Press, 1998.

[106] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. A. Solla, T. K. Leen, and K. Müller. 2000, pp. 1057–1063.

[107] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112.1 (1999), pp. 181–211.

[108] Csaba Szepesvári and Michael L. Littman. *Generalized Markov Decision Processes: Dynamic-programming and Reinforcement-learning Algorithms*. Tech. rep. University of Szeged, 1997.

[109] Daniel Szer, François Charpillet, and Shlomo Zilberstein. "MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs". In: *21st Conference on Uncertainty in Artificial Intelligence*. Ed. by Max Chickering, Fahiem Bacchus, and Tommi Jaakkola. 2005, pp. 576–590.

[110] Alejandro Toriello, William B. Haskell, and Michael Poremba. "A Dynamic Traveling Salesman Problem with Stochastic Arc Costs". In: *Operations Research* 62.5 (2014), pp. 1107–1125.

[111] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Ed. by Paolo Toth and Daniele Vigo. Society for Industrial and Applied Mathematics, 2002.

[112] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. 2017, pp. 5998–6008.

[113] M. Veres and M. Moussa. "Deep Learning for Intelligent Transportation Systems: A Survey of Emerging Trends". In: *IEEE Transactions on Intelligent Transportation Systems* 21.8 (2020), pp. 3152–3168.

[114] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. *FeUdal Networks for Hierarchical Reinforcement Learning*. arXiv. 2017. eprint: 1703.01161 (cs.AI).

[115] Daniele Vigo. *VRPLIB*. online. 1995. URL: http://or.dei.unibo.it/library/vrplib-vehicle-routing-problem-library.

[116] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. "Pointer Networks". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. 2015, pp. 2692–2700.

[117] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3 (1992), pp. 229–256.

[118] Feng Wu, Shlomo Zilberstein, and Nicholas R. Jennings. "Monte-Carlo Expectation Maximization for Decentralized POMDPs". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*. Ed. by Sebastian Thrun, Francesca Rossi, Carla Gomes, and Umberto Grandi. 2013, pp. 397–403.

FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

NOM : BONO

DATE de SOUTENANCE : 28/10/2020

Prénoms : Guillaume

TITRE : Deep Multi-Agent Reinforcement Learning for Dynamic and Stochastic Vehicle Routing Problems

NATURE : Doctorat

Numéro d'ordre : 2020LYSEI096

Ecole doctorale : InfoMaths (ED512)

Spécialité : Informatique

RESUME : La planification de tournées de véhicules dans des environnements urbains denses est un problème difficile qui nécessite des solutions robustes et flexibles. Les approches existantes pour résoudre ces problèmes de planification de tournées dynamiques et stochastiques (DS-VRPs) sont souvent basés sur les mêmes heuristiques utilisées dans le cas statique et déterministe, en figeant le problème à chaque fois que la situation évolue. Au lieu de cela, nous proposons dans cette thèse d'étudier l'application de méthodes d'apprentissage par renforcement multi-agent (MARL) aux DS-VRPs en s'appuyant sur des réseaux de neurones profonds (DNNs). Plus précisément, nous avons d'abord contribuer à étendre les méthodes basées sur le gradient de la politique (PG) aux cadres des processus de décision de Markov (MDPs) partiellement observables et décentralisés (Dec-POMDPs). Nous avons ensuite proposé un nouveau modèle de décision séquentiel en relâchant la contrainte d'observabilité partielle que nous avons baptisé MDP multi-agent séquentiel (sMMDP). Ce modèle permet de décrire plus naturellement les DS-VRPs, dans lesquels les véhicules prennent la décision de servir leurs prochains clients à l'issu de leurs précédents services, sans avoir à attendre les autres. Pour représenter nos solutions, des politiques stochastiques fournissant aux véhicules des règles de décisions, nous avons développé une architecture de DNN basée sur des mécanismes d'attention (MARDAM). Nous avons évalué MARDAM sur un ensemble de bancs de test artificiels qui nous ont permis de valider la qualité des solutions obtenues, la robustesse et la flexibilité de notre approche dans un contexte dynamique et stochastique, ainsi que sa capacité à généraliser à toute une classe de problèmes sans avoir à être réentraînée. Nous avons également développé un banc de test plus réaliste à base d'une simulation micro-traffic, et présenté une preuve de concept de l'applicabilité de MARDAM face à une variété de situations différentes.

MOTS-CLÉS : Problème de Planification de Tournées Dynamique et Stochastique (DS-VRP),
Apprentissage par Renforcement Profond (Deep RL), Mécanismes d'Attention, Systèmes Multi-Agent (MAS)

Laboratoire (s) de recherche : CITI

Directeur de thèse : Olivier SIMONIN

Président de jury : René MANDIAU

Composition du jury : René MANDIAU, François CHARPILLET, Romain BILLOT, Aurélie BEYNIER, Christian WOLF, Olivier SIMONIN, Jilles DIBANGOYE, Laëtitia MATIGNON.