POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Entrepôts autonomes à l'ère du e-commerce : apprentissage automatique pour la prise de décision en temps réel

ADRIEN RIMÉLÉ
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques

Août 2021

© Adrien Rimélé, 2021.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Entrepôts autonomes à l'ère du e-commerce : apprentissage automatique pour la prise de décision en temps réel

présentée par **Adrien RIMÉLÉ**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Issmaïl EL HALLAOUI, président
Louis-Martin ROUSSEAU, membre et directeur de recherche
Michel GAMACHE, membre et codirecteur de recherche
Michel GENDREAU, membre et codirecteur de recherche
Antoine LEGRAIN, membre
Barrett THOMAS, membre externe

REMERCIEMENTS

Je souhaite remercier un certain nombre de personnes sans lesquelles ce projet de doctorat n'aurait pu être mené à bien, ou qui, par leur présence, ont rendu cette expérience particulièrement plaisante et enrichissante.

Je souhaite tout d'abord exprimer ma profonde gratitude à mon directeur de recherche, Prof. Louis-Martin Rousseau, dont la vision et expertise dans de nombreux domaines est sans limites. Je remercie également mes deux co-directeurs, Prof. Michel Gamache et Prof. Michel Gendreau, pour leur supervision et leurs suggestions toujours éclairées. En plus d'avoir dirigé ma maîtrise lorsque je suis arrivé au Canada, Prof. Gamache m'a offert l'opportunité de donner ses séances de laboratoires de recherche opérationnelle, je lui suis extrêmement reconnaissant tant cette expérience fut enrichissante.

Je tiens également à remercier l'organisme de financement Mitacs qui a financé mon premier stage de recherche chez JDA software. Un grand merci à l'équipe RO de JDA avec laquelle j'ai eu la chance de travailler, en particulier Marc Brisson, manager de l'équipe, mais aussi Éric, Philippe, Thierry et Vincent avec qui j'ai beaucoup appris et passé d'excellents moments. J'ai également eu le plaisir de réaliser un second stage de recherche, cette fois à Element AI. Je remercie tout d'abord Sara Morin, manager de l'équipe pour son suivi toujours très attentionné. Merci à tous les membre de l'équipe : Adham, Ben, Dominique, Éric, Étienne, Masoud, Mathieu, Patrick, Philippe et Thierry, pour ces bons moments passés ensemble et pour m'avoir inclus aussi chaleureusement dans vos activités professionnelles et sociales.

Je me permets de consacrer un paragraphe pour remercier spécifiquement Philippe Grangier. Philippe a supervisé mes deux stages au sein de JDA et Element AI et a continué à suivre mes avancées par la suite. Successivement superviseur puis mentor et ami, je n'aurais pu imaginer un meilleur encadrement. Merci !

Je remercie également l'ensemble des membres de mes laboratoires de recherche, le CIRRELT et HANALOG, pour tous ces bons moments et échanges constructifs. Merci tout particulièrement à Mahdis et Peyman avec qui j'ai eu le plaisir de partager mon bureau toutes ces années. Notre amitié se place au plus haut dans ce qu'a pu m'offrir ce doctorat : *motshakeram*, merci.

Un grand merci à mes amis Alexandre et Lancelot pour être restés si présents, même en étant si loin. Ils se sont même dévoués pour relire une partie de ce manuscrit !

Évidemment, je remercie du fond du cœur mes parents, à qui je dois tout, et qui m'ont toujours encouragé et soutenu dans la poursuite de mes études, aussi longtemps que je le

souhaitais.

Enfin, un énorme merci à Fernanda, ma compagne, pour m'avoir accompagné toutes ces années. Tu m'as tant apporté à bien des égards, toujours soutenu avec un sourire infaillible. Nous avons fini cette aventure avec le bonheur d'avoir Lucas et Mateo, maintenant âgés de treize mois, et tous les défis qui les accompagnent. Partenaire idéale dans une situation exceptionnelle, cette thèse n'aurait pas été possible sans toi.

RÉSUMÉ

La récente et rapide croissance du commerce et ligne a fondamentalement changé le secteur de la vente au détail et, plus généralement, la façon dont les individus consomment. Cette industrie fait face à une grande compétitivité avec une grande volatilité de sa clientèle. En cherchant constamment à réduire ses délais de livraison tout en minimisant les coûts, les chaînes logistiques font face à un défi de taille pour satisfaire une demande grandissante et s'adapter à des tendances qui évoluent constamment. Cette thèse considère un nouveau type d'entrepôt de stockage automatique, appelé *centre de distribution robotisé*, qui est particulièrement bien adapté aux spécificités du commerce en ligne.

Dans un centre de distribution robotisé, une flotte de robots récupère et entrepose des étagères remplies d'articles dans l'espace de stockage. Ces étagères sont apportées à des stations de cueillette où des opérateurs humains sélectionnent les articles pour satisfaire les commandes. En pratique, un robot réalise une succession continue de cycles de commandes doubles. Ces cycles correspondent à une première tâche de stockage, suivi immédiatement d'une tâche de récupération, puis d'une durée d'attente à une station. Ce processus engendre une multitude de problèmes de décision tels que l'allocation d'emplacements de stockage, l'ordonnancement des commandes, l'affectation d'une station de cueillette et autres. De plus, la promesse d'une livraison toujours plus rapide force les vendeurs à considérer une nouvelle commande dès que celle-ci est placée. Pour cette raison, ainsi que la nature dynamique du système d'entreposage, la prise de décision doit être réalisée en temps réel dans un environnement incertain.

Cette thèse s'organise autour de trois articles publiés ou soumis dans des journaux scientifiques. Compte tenu du manque habituel de formalisation des problèmes de décision en temps réel, le premier article propose un modèle de programme dynamique stochastique qui modélise les décisions opérationnelles dans un centre de distribution robotisé. L'objectif de ce modèle est de formaliser les opportunités d'optimisation dans ce système afin de permettre aux chercheurs de développer de nouvelles approches dans un environnement bien défini. Reproduit par un simulateur à événements discrets, ce modèle est illustré par une étude de simulation qui vise à comparer les règles de décision standards d'allocation d'emplacements de stockage : un stockage aléatoire, sélection de l'emplacement disponible le plus proche, un stockage par classes et la politique gloutonne de temps d'accès et de transition le plus court.

Le second article présente une méthode d'apprentissage d'une politique de stockage par zones. Un élément essentiel des centres de distribution robotisés est leur grande modularité de stockage. Lorsqu'un robot replace une étagère, il peut sélectionner n'importe quel emplacement

disponible. Étant donné le nombre important de robots opérant simultanément, cela crée de nombreuses opportunités. Cette modularité permet d'adapter la configuration de l'entrepôt à de nouvelles tendances de la demande et de créer des cycles les plus efficaces possibles. Cet article utilise une méthode d'apprentissage par renforcement qui vise à minimiser le temps de déplacement des robots. Les décisions sont prises dans un processus de décision markovien partiellement observable où l'état du système est représenté par de l'information sur la configuration actuelle de l'espace de stockage, la prochaine commande à traiter et des statistiques sur les distributions d'arrivée des commandes. Basé sur cette représentation, un agent de Deep Q-learning est utilisé pour apprendre à quelle zone de l'entrepôt chaque étagère devrait être dynamiquement affectée. De plus, une stratégie d'exploration de trajectoires est proposée afin de tirer profit des informations sur les commandes déjà révélées à un instant donné. Avec une étude de simulation, la méthode proposée, incluant l'exploration de trajectoires, génère des résultats qui améliorent de 7.6% la meilleure règle standard de décision en termes de temps de déplacement.

Le troisième article propose une autre approche d'apprentissage pour sélectionner dynamiquement les emplacements de stockage exacts (au lieu de zones) en utilisant des méthodes de recherche d'arbres. Une recherche arborescente Monte-Carlo est employée hors ligne pour accumuler de l'expérience de haute qualité. Cette recherche d'arbre étant particulièrement coûteuse en temps de calcul, elle ne peut être déployée en temps réel. Cependant, un réseau de neurones peut être utilisé pour apprendre de cette expérience avec une certaine représentation d'état du système (les emplacements libres et les étagères requises sont représentées par leurs coordonnées). Ce réseau est ensuite employé comme prédicteur d'actions dans plusieurs nouvelles politiques de stockage, soit tel quel, soit dans des stratégies d'exploration de trajectoires ou de recherches d'arbres supervisées. Le niveau de performance obtenu est constamment supérieur aux règles de décision de la littérature et dépend du temps de calcul alloué à la prise de décision. Par exemple, la meilleure politique de recherche d'arbre supervisée améliore la meilleure règle de décision de 13.7%.

ABSTRACT

The growth of e-commerce has changed the way people consume and the business-to-consumer retail market segment. The goal of increasing delivery speed while remaining cost-effective poses significant new challenges for supply chains as they race to satisfy the growing and fast-changing demand. This thesis considers a recent automated warehouse called Robotic Mobile Fulfillment System (RMFS), particularly well-suited for e-commerce operations.

In an RMFS, a fleet of small robots retrieves and stores shelves of items in the storage area. These robots bring the shelves to picking stations, where a human operator picks the required items to fulfil orders. The life of a robot is a succession of dual command cycles where a storage task is immediately followed by a retrieval task to finally wait in line at a picking station. Such cycles involve numerous decisions, from storage allocation to order sequencing, picking station assignments, and others. Also, online retailers promise speedy deliveries, which requires that new orders be included in the set of requests to fulfil as soon as they are known. For this reason, and because of the very dynamic nature of the robots' cycles, decision-making needs to be done in real-time, in an uncertain environment.

This thesis is articulated around three articles, published, or submitted to scientific journals. First, and because such a real-time problem often lacks a formal description, we propose a mathematical framework that models decision-making in an RMFS as a stochastic dynamic program. This model's objective is to formalize optimization opportunities to allow researchers to develop more advanced methods in a well-defined environment. Embedded in a discrete event simulator, this model is illustrated by simulations to compare against standard storage decision rules, including a random storage policy, a closest open location policy, a class-based storage policy, and the shortest leg (greedy) policy.

In a second phase, we present a method that learns a zone-based storage policy. A distinguishing feature of an RMFS is its great storage modularity. Indeed, when a robot returns a shelf to the storage area, it can select any available location. Because of the large number of simultaneously operating robots, there are plenty of available options. This flexibility allows the warehouse's layout to adapt to new demand patterns and create efficient cycles. We propose to tackle this storage allocation sub-problem by using Deep Reinforcement Learning to minimize the robots' average travel time. The decision-making process is first formulated as a Partially Observable Markov Decision Process where information about the current layout of the warehouse, the next order in line, and the demand distribution is available. Based on this state representation and a discrete event simulator, a Deep Q-learning agent is used to

learn to which zone a shelf should be dynamically assigned. Additionally, we propose a rollout strategy to enhance our method by leveraging more information about already revealed orders at a given time step. Using simulations to compare our method to the best standard decision rule (shortest leg) showed average performance gains of 7.6% in travel time.

Finally, and motivated by the good results of the rollout strategy, a different learning approach is proposed to dynamically select exact storage locations (instead of zones) using tree search methods. This approach first consists of running offline an efficient yet computation costly Monte Carlo Tree Search method to generate a high-quality experience. This experience is later learned by a neural network with a proper coordinates-based features representation. The obtained neural network is used as an action predictor in several new storage policies, either as it is or in rollout and supervised tree search methods. The supervised tree search method explores the tree's nodes most likely to be visited by the policy network. The obtained performance levels depend on the computation time available at a decision step and are highly competitive compared to real-time decision rules from the literature. For instance, the best-supervised tree search instance improves by 13.7% the shortest leg policy.

TABLE DES MATIÈRES

REMERCIEMENTS iii

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES ix

LISTE DES TABLEAUX xii

LISTE DES FIGURES xiii

LISTE DES SIGLES ET ABRÉVIATIONS xiv

CHAPITRE 1 INTRODUCTION 1
- 1.1 Définitions et concepts de base 1
- 1.2 Éléments de la problématique 3
- 1.3 Objectifs de recherche 5
- 1.4 Plan de la thèse 5

CHAPITRE 2 REVUE DE LITTÉRATURE 7
- 2.1 Systèmes automatisés de stockage et de récupération 7
  - 2.1.1 Description 7
  - 2.1.2 Politiques de stockage 9
  - 2.1.3 Stockage dédié 11
  - 2.1.4 Stockage statique 13
  - 2.1.5 Stockage partagé 14
- 2.2 Centres de distribution robotisés 16
  - 2.2.1 Modèles analytiques 16
  - 2.2.2 Ordonnancement des commandes 17
  - 2.2.3 Allocation d'emplacements de stockage 18
  - 2.2.4 Autres thématiques 19

CHAPITRE 3 DÉMARCHE ET ORGANISATION DE LA THÈSE 20

CHAPITRE 4 ARTICLE 1 : ROBOTIC MOBILE FULFILLMENT SYSTEMS : A MATHEMATICAL MODELLING FRAMEWORK FOR E-COMMERCE APPLICATIONS . . . 23
4.1 Introduction . . . 24
4.2 Robotic Mobile Fulfillment Systems . . . 25
4.3 Literature review . . . 26
4.4 Formulation . . . 28
4.4.1 Modelling framework . . . 28
4.4.2 Sets . . . 30
4.4.3 Parameters . . . 31
4.4.4 State variables . . . 32
4.4.5 Exogenous information . . . 35
4.4.6 Decisions . . . 36
4.4.7 Transition function . . . 37
4.4.8 Cost function . . . 41
4.5 Simulation study . . . 44
4.5.1 Assumptions . . . 44
4.5.2 Storage allocation baselines . . . 45
4.5.3 Simulation study . . . 46
4.6 Conclusions and future work . . . 49

CHAPITRE 5 ARTICLE 2 : E-COMMERCE WAREHOUSING : LEARNING A STORAGE POLICY . . . 51
5.1 Introduction . . . 52
5.1.1 Warehousing in e-commerce . . . 52
5.1.2 Robotic Mobile Fulfillment System . . . 52
5.1.3 Contributions and paper structure . . . 53
5.2 Problem definition and literature . . . 54
5.2.1 Decision-making framework . . . 54
5.2.2 Storage policies . . . 55
5.2.3 RMFS-related work . . . 56
5.3 Methodology . . . 58
5.3.1 Storage zones definition . . . 58
5.3.2 System representation . . . 59
5.3.3 Reinforcement learning method . . . 60
5.3.4 Implementation details . . . 62

5.3.5 Look-ahead rollout strategy 63
5.4 Simulation study 65
5.4.1 Parameters 65
5.4.2 Results 67
5.5 Conclusions 70

CHAPITRE 6 ARTICLE 3 : SUPERVISED LEARNING AND TREE SEARCH FOR REAL-TIME STORAGE ALLOCATION IN ROBOTIC MOBILE FULFILLMENT SYSTEMS 72
6.1 Introduction 72
6.2 Literature review 74
6.2.1 Analytical models 74
6.2.2 Scheduling and task allocation 75
6.2.3 Storage allocation 76
6.3 Problem definition 77
6.4 Methodology 78
6.4.1 Monte Carlo Tree Search algorithm 78
6.4.2 Learning a policy 79
6.4.3 Enhancement of the Learned Policy 82
6.4.4 Supervised Tree Search algorithm 83
6.5 Simulation study 86
6.5.1 Preliminary results 86
6.5.2 Results 88
6.6 Conclusion 90

CHAPITRE 7 DISCUSSION GÉNÉRALE 97

CHAPITRE 8 CONCLUSION ET RECOMMANDATIONS 99
8.1 Synthèse des travaux 99
8.2 Limitations et améliorations futures 100

RÉFÉRENCES 102

LISTE DES TABLEAUX

Table 4.1 Travelling times 48
Table 4.2 Full cycle times 49
Table 5.1 Average travelling times t(s) and performance gains g(%) depending on distribution skewness parameter value s - without rollouts 68
Table 5.2 Performance gains g(%) depending on distribution skewness parameter value s and horizon h - with look-ahead rollouts 70
Table 6.1 Results of the methods without learning 92
Table 6.2 Results of the methods learned from MCTS(5) 93
Table 6.3 Results of the methods learned from MCTS(10) 94
Table 6.4 Results of the methods learned from MCTS(20) 95
Table 6.5 Results of the methods learned from MCTS(30) 96

LISTE DES FIGURES

Figure 4.1 Decision tree 31
Figure 4.2 Possible decisions when a robot is located at a picking station 37
Figure 4.3 Illustration of the cost components in the complete cycle time 43
Figure 4.4 Simulation study - storage area 48
Figure 5.1 Dual-command cycle (left) and opportunistic task (right) 55
Figure 5.2 Concentric class-based storage layout (left) vs. proposed zones layout (right) 59
Figure 5.3 Revealed orders at decision time 64
Figure 5.4 Look-ahead rollout strategy 65
Figure 5.5 Simulation study - Plan view of the storage area 66
Figure 5.6 Typical instance of a training curve (here for s=0.6) 68
Figure 6.1 MCTS 80
Figure 6.2 Simple features representation illustration 81
Figure 6.3 Supervised Tree Search 84
Figure 6.4 Plan view of the storage area 87
Figure 6.5 Correlation between the network's accuracy and the policies' performance 88
Figure 6.6 Policies' performance 89
Figure 6.7 Learned policies' performance 89

LISTE DES SIGLES ET ABRÉVIATIONS

AS/RS	Automated Storage and Retrieval System (système automatique de stockage et de récupération)
COI	Cube per Order Index (indice cubique par commande)
COL	Closest Open Location (emplacement libre le plus proche)
DC	Dual-Command (commande double)
FCFS	First-Come First-Served (premier arrivé, premier servi)
I/O point	Input/Output location (point d'entrée-sortie)
KPI	Key Performance Indicator (indicateur clé de performance)
LP	Learned Policy (politique apprise)
SR	Storage and Retrieval (stockage et récupération)
MCTS	Monte Carlo Tree Search (recherche arborescente Monte-Carlo)
MDP	Markov Decision Process (processus de décision markovien)
MIP	Mixed Integer Programming (programmation mixte en nombres entiers)
NN	Nearest Neighbour (voisin le plus proche)
PO-MDP	Partially Observable Markov Decision Process (processus de décision markovien partiellement observable)
RL	Reinforcement Learning (apprentissage par renforcement)
RMFS	Robotic Mobile Fulfillment System (centre de distribution robotisé)
SDT	Shortest Due Time (temps d'échéance le plus court)
SL	Shortest Leg (temps d'accès et de transition le plus court)
STS	Supervised Tree Search (recherche d'arbre supervisée)
STT	Shortest Total Time (temps total le plus court)

CHAPITRE 1 INTRODUCTION

Ce travail de recherche porte sur la prise de décision en temps réel dans un nouveau type d'entrepôt automatisé adapté au e-commerce. Dans cet entrepôt, appelé *centre de distribution robotisé*, une flotte de robots récupère des étagères dans l'espace de stockage afin de les apporter à des stations de cueillette où des opérateurs traitent les commandes. Les caractéristiques des commandes en ligne, ainsi que celles intrinsèques à ces entrepôts, résultent en un système dynamique et complexe qui doit continuellement s'adapter à la demande. Entre autres problèmes de décision, l'allocation d'emplacements de stockage présente une grande opportunité d'optimisation. En effet, après chaque opération de cueillette, un robot peut réallouer une étagère à n'importe quel emplacement libre. Cette modularité permet non seulement au système de s'adapter aux tendances de la demande, mais également de saisir des opportunités à court terme afin de minimiser, par exemple, la durée de déplacement des robots et ainsi maximiser la productivité.

1.1 Définitions et concepts de base

La croissance rapide et récente du commerce en ligne a révolutionné le secteur de la vente au détail des entreprises aux particuliers (Business-to-Consumer en anglais) et, plus globalement, la façon dont les individus consomment. Sur le marché américain, cette croissance s'est maintenue à un niveau élevé et relativement constant de +15% par an depuis une décennie. L'année 2020, exceptionnelle dans tous les domaines, a généré une croissance record de +44%. Aujourd'hui, le commerce en ligne représente 21.3% de l'ensemble de la vente au détail, pour un total de 861 milliards de dollars aux États-Unis [1].

Le commerce en ligne présente des caractéristiques uniques. Tout d'abord, il implique un volume significatif de petites commandes, ce qui force les vendeurs à maintenir un inventaire très important. Boysen et al. [2] mentionne par exemple que le nombre d'articles par commande chez Amazon est de 1.6. Le commerce en ligne fait également face à une grande incertitude de la demande, avec des tendances qui changent très régulièrement. De plus, les commandes doivent généralement être satisfaites très rapidement [2–4]. Amazon offre par exemple à ses clients un service *Prime* qui, dans certains cas, garantit une livraison le jour même. Avec une clientèle qui choisit ses articles exclusivement par le biais d'une description, la vente en ligne génère un grand nombre de retours qui doivent être réintroduits dans la chaîne logistique. Aussi, avec des alternatives facilement disponibles aux clients, les vendeurs en ligne font face à une grande compétitivité. Il est d'ailleurs démontré que la relation entre

la performance logistique et la fidélité de la clientèle est bien plus forte que celle d'autres secteurs [5].

Pour répondre à la demande croissante et aux caractéristiques du commerce en ligne, un nouveau type d'entrepôt automatisé, appelé centre de distribution robotisé, a récemment vu le jour (ou Robotic Mobile Fulfillment System, RMFS, en anglais). Le premier RMFS a été créé par Kiva Systems en 2006, entreprise rachetée par Amazon en 2012 et renommée Amazon Robotics [4, 6, 7]. Depuis, d'autres systèmes similaires concurrents ont vu le jour tels que les robots Zhu Que d'Alibaba, les Quicktrons de Huawei, les Open Shuttles de Knapp ou le système CarryPick de Swisslog [4, 8]. Il est estimé que les ventes de ces systèmes robotisés génèreront un chiffre d'affaires de $30.8Md en 2022, soit environ 900K unités de robots [9].

Un RMFS est un système *articles-vers-cueilleur* (parts-to-picker) dans lequel une flotte de robots déplace des étagères remplies d'articles au sein de l'espace de stockage. Plus précisément, un robot réalise une série de *cycles de commandes doubles* qui consistent en une succession de tâches de stockage et de récupération afin d'apporter des étagères à des opérateurs humains localisés à des stations de cueillette. Ainsi, lorsqu'un opérateur a terminé sa tâche de cueillette, le robot doit replacer l'étagère dans l'espace de stockage. Par la suite, il se rend directement à l'étagère associée à la prochaine commande pour l'apporter jusqu'à une station de cueillette. À cette dernière, il est possible que le robot doive se placer dans une file d'attente, en attendant que d'autres robots soient libérés. Un élément d'intérêt dans cette thèse est la possibilité pour un robot de sélectionner n'importe quel emplacement de stockage disponible à un instant donné. Le système peut ainsi s'adapter aux nouvelles tendances de la demande ou saisir des opportunités pour une plus grande productivité. En pratique un robot se déplace en suivant des marqueurs au sol disposés en forme de grille. Il est intéressant de noter que lorsqu'un robot est chargé, il doit se déplacer dans les allées. En revanche, lors des transitions, le robot se déplace à vide, il peut ainsi passer sous les étagères entreposées pour se rendre plus directement à sa destination et éviter le trafic. De plus amples détails sur le fonctionnement des RMFSs peuvent être trouvés dans les références suivantes : Wurman et al. [6], D'Andrea et Wurman [10], Enright et Wurman [11], ou encore Azadeh et al. [7].

Les RMFSs présentent plusieurs avantages par rapport aux systèmes manuels ou aux systèmes automatisés de stockage et de récupération plus conventionnels (Automated Storage and Retrieval Systems, AS/RSs) [6]. Tout d'abord, un robot peut accéder à l'ensemble de l'entrepôt et n'importe quelle station de cueillette (ou de réapprovisionnement) peut être desservie. Moins de main-d'œuvre est nécessaire aux opérations de cueillette, il y a moins de dépendances entre les opérateurs, et donc de possibles délais. Cela a également pour effet de faciliter le suivi des responsabilités et de réduire les erreurs. Étant donné le nombre important

de robots opérant simultanément, il n'y a pas de point de rupture. Le système présente une excellente modularité qui lui permet d'adapter sa configuration en continue pour satisfaire au mieux la demande. Enfin, ce système est facilement extensible, ce qui permet d'adapter la taille de l'installation à la croissance d'une entreprise.

1.2 Éléments de la problématique

Compte tenu de la pression exercée sur les RMFSs pour satisfaire les commandes aussi vite que possible, les nouvelles commandes doivent être considérées immédiatement. Banker [12] mentionne une *logique de traitement des commandes en continue* (ou *order streaming logic*) qui consiste à "traiter les commandes dès qu'elles sont reçues" ("drops these orders [...] to the floor as soon as they are received"). Cette logique limite fortement le regroupement des commandes qui consisterait à fixer un bloc de commandes, afin de planifier son traitement, avant de passer au bloc suivant. Cette considération suggère une prise de décision qui devrait être faite en temps réel. La réalisation simultanée de cycles de commandes doubles par les robots présente une grande complexité. Par exemple, les emplacements disponibles ainsi que les emplacements des étagères (et donc les temps d'accès) dépendent des décisions prises continuellement. Ceci, couplé au déplacement des robots et au temps qu'ils passent dans les files d'attente aux stations, crée un système extrêmement dynamique. Quelque soit le type d'approche utilisé pour la prise de décision, un tel système doit être évalué au travers d'un simulateur qui permette de reproduire fidèlement les diverses opérations d'entreposage.

Un RMFS présente une multitude de problèmes de décision. Merschformann et al. [13] identifient par exemple les problèmes suivants : l'ordonnancement du traitement des commandes ; l'affectation d'une étagère à une commande ; l'allocation d'un emplacement de stockage à une étagère ; la sélection d'une station pour une tâche de réapprovisionnement ou encore d'une étagère à réapprovisionner. À ces problèmes peuvent s'ajouter le choix d'une station de cueillette pour le traitement d'une commande, l'ordonnancement des tâches de réapprovisionnement, ou la possibilité pour un robot de court-circuiter l'espace de stockage en réalisant une tâche alternative à une tâche de stockage, appelée *tâche opportuniste* dans cette thèse.

La possibilité de réaliser une tâche opportuniste se présente lorsqu'un robot est libéré par un opérateur à une station. Au lieu de réaliser une tâche de stockage, le robot peut décider de réutiliser immédiatement son étagère pour traiter une nouvelle commande afin d'économiser du temps de déplacement. Pour cela, il faut évidemment qu'une commande requière un article présent dans l'étagère. Plusieurs options sont alors possibles : le robot peut rester en place à la même station, afin de traiter la nouvelle commande immédiatement ; il peut aussi se replacer dans une file d'attente de la même ou de n'importe quelle autre station.

Dans cette thèse, il est considéré que toutes les prises de décision associées aux différents sous-problèmes se présentent lorsqu'un robot devient disponible, typiquement juste après une opération de cueillette (ou de réapprovisionnement) ou après le dépôt d'une étagère à son emplacement de stockage. Ceci correspond d'ailleurs à la notion d'*événements* définie dans un simulateur à événements discrets, utilisé pour évaluer la performance du système. De plus, quelque soit les décisions qui doivent être prises, elles sont considérées *non-préemptives*, dans le sens où une nouvelle décision ne doit pas venir altérer une décision antérieure. Par exemple, si un robot est présentement en déplacement vers un emplacement de stockage, ce dernier ne peut pas être réalloué à un autre robot (ce qui forcerait le premier robot à changer de tâche), même si cela générerait une économie. Enfin, d'autres problèmes de décision plus spécifiques à la robotique tels que le routage des robots, les opérations de maintenance, de chargement des batteries et autres, ne sont pas considérés dans ce travail.

Le problème d'allocation d'emplacements de stockage reçoit une attention particulière dans cette thèse. Comme mentionné plus haut, lorsqu'un robot portant une étagère est libéré à une station de cueillette, un nouvel emplacement où entreposer l'étagère doit être sélectionné. Un objectif habituel est alors de minimiser le temps de déplacement moyen des cycles de commandes doubles réalisés par les robots. Ce temps est une borne inférieure sur le temps de cycle total moyen (incluant l'attente à une station), qui à son tour est inversement proportionnel au flux de production maximum. Autrement dit, minimiser le temps de déplacement des robots permet d'augmenter le flux de production maximum de l'entrepôt.

Il est commun dans les systèmes d'entreposage que la demande ne soit pas uniforme mais fortement biaisée vers un nombre limité de conteneurs ou d'articles, représentant la majorité des commandes. Intuitivement, il existe donc un compromis entre positionner des étagères souvent demandées près des stations et réaliser des cycles les plus courts possibles. En pratique, cette prise de décision en temps réel est gérée par des règles de décision qui utilisent précisément ces deux notions. Ainsi, inspirés par les systèmes automatiques à cycles de commandes simples (allers-retours entre l'espace de stockage et la station), les politiques basées sur le taux de demande (nombre de requêtes par unité de temps) ordonnent les étagères par taux de demande croissants et les allouent à des emplacements ou des classes (groupes d'emplacements) de plus en plus loin des stations. À l'inverse, la politique gloutonne de *temps d'accès et de transition* le plus court, appelée *Shortest Leg* (SL) en anglais, est exclusivement concentrée sur la durée du cycle immédiat. Compte tenu du nombre de robots opérant simultanément dans un RMFS, une multitude d'emplacements de stockage est disponible à un instant donné. Cela semble donner un fort avantage aux politiques de type SL dans le cas général. Ces sujets, ainsi que d'autres approches qui supposent un regroupement de commandes, seront étudiés plus en détail dans la revue de littérature au chapitre 2. Cette

thèse explore l'apprentissage de politiques de stockage visant à tirer profit de ce compromis. Quel que soit le type d'approche développé (type de modélisation, méthode de résolution, technique d'apprentissage, etc.), il est essentiel pour une politique d'être parfaitement déployable en temps réel. Elle doit être capable de réagir en continue à un flux de nouvelles informations et d'avoir un temps de calcul en déploiement très limité.

1.3 Objectifs de recherche

L'objectif général de la thèse est de proposer une approche de résolution adaptée à la prise de décision en temps réel dans un centre de distribution robotisé, système hautement dynamique et incertain. Plus précisément, deux objectifs spécifiques sont formulés de la manière suivante :

1. Formaliser le problème de prise de décision en temps réel dans un centre de distribution robotisé.
2. Proposer de nouvelles politiques de stockage plus performantes que les règles de décision couramment employées. Ces politiques doivent être à la fois performantes et parfaitement déployables en temps réel.

Le premier objectif cherche à combler un manque habituel rencontré dans des problématiques de prise de décision en temps réel. Dans ce contexte, un modèle mathématique permet d'isoler les décisions opérationnelles et de formaliser leur influence sur le système par le biais de contraintes, de fonctions de transition et d'objectifs. Le deuxième objectif vise à proposer des politiques alternatives aux règles de décision traditionnelles. Compte tenu de la dynamique du système, de la séquentialité de la prise de décision, mais également de la constance de l'environnement dans lequel il évolue, l'apprentissage automatique se présente comme une approche intéressante pour une prise de décision rapide et éclairée. De plus, la prise de décision en temps réel présente généralement un budget computationnel correspondant à un intervalle de temps raisonnable pour chaque décision. Si ce budget le permet, il peut être envisagé de déployer une méthode de recherche d'arbre de décision qui utilise une politique apprise (prédicteur) pour simuler la réponse du système à une séquence de décisions.

1.4 Plan de la thèse

La thèse s'organise autour de trois articles de recherche publiés ou soumis dans des journaux scientifiques. Tout d'abord, une revue de littérature est présentée au chapitre 2. Cette revue étudie les travaux de recherche menés dans le domaine des entrepôts automatiques de stockage et de récupération, ainsi que des études, plus récentes, spécifiques aux centres de distribution robotisés. Le chapitre 3 décrit l'organisation générale du projet de recherche, ce qui a motivé

les différents articles et le lien qui les relie. Le chapitre 4 présente le premier article portant sur la modélisation de la prise de décision en temps réel dans les centres de distribution robotisés. Le chapitre 5 présente le deuxième article qui utilise un apprentissage par renforcement pour apprendre une politique de stockage en temps réel. Cet apprentissage vise à minimiser le temps de cycle moyen des robots dans un processus de décision markovien partiellement observable avec de l'information sur l'état actuel de l'entrepôt, la prochaine commande et des statistiques associées à la demande. Le troisième article, présenté au chapitre 6, a également pour objectif d'apprendre une politique de stockage, mais en utilisant de l'apprentissage supervisé et des techniques d'exploration d'arbre de recherche afin de tirer profit d'une liste de commandes révélées. Le chapitre 7 discute de l'ensemble des travaux de recherche et des principaux résultats obtenus. Enfin, le chapitre 8 synthétise les travaux réalisés, souligne des limitations et propose des pistes de future recherche.

CHAPITRE 2 REVUE DE LITTÉRATURE

Bien que les centres de distribution robotisés (Robotic Mobile Fulfillment Systems, RMFSs) soient des systèmes d'entreposage relativement récents, la littérature sur le sujet remonte aux années 1960 et au domaine des systèmes automatisés de stockage et récupération plus conventionnels (Automated Storage and Retrieval Systems, AS/RSs). Ce type d'entrepôt automatique continue à être utilisé de nos jours et partage de nombreuses caractéristiques avec les RMFSs, particulièrement dans sa configuration la plus répandue, à savoir un AS/RS à charge-unique et allée-dépendante. Cette revue de littérature va donc s'articuler autour de deux axes principaux : les travaux de recherche concernant les AS/RSs et les travaux, plus récents, développés spécifiquement pour les RMFSs.

2.1 Systèmes automatisés de stockage et de récupération

2.1.1 Description

En comparaison avec un système d'entreposage manuel, un AS/RS est un système *"parts-to-picker"* qui n'implique pas de déplacement d'un opérateur humain au sein de l'espace de stockage. Roodbergen et Vis [14] présentent une revue de littérature complète sur les AS/RSs. De nombreuses configurations existent mais la version la plus habituelle est un système à charge unique et allée-dépendante. Dans celui-ci, une machine de Stockage et Récupération (SR) déplace les conteneurs stockés dans des racks verticaux le long d'une allée. Il y a une seule machine SR par allée et elle ne peut en desservir d'autres. Lorsque la machine a récupéré un conteneur, elle l'apporte au point d'*Input/Output* (I/O) du système. À ce point de transition, différentes configurations sont possibles : un opérateur humain peut y être placé pour réaliser une tâche de cueillette avant que le conteneur ne retourne dans les racks (système miniload) ou un convoyeur peut simplement emporter le conteneur au complet dans une autre zone de l'entrepôt.

La machine SR réalise une succession d'allers-retours entre le point I/O et l'espace de stockage. Ils définissent des cycles qui peuvent être de deux types : cycles de commandes simples (*single command cycles*) ou cycles de commandes doubles (*dual command cycles*). Dans un cycle de commande simple, la machine se contente de réaliser une tâche (de stockage ou de récupération), et revient immédiatement au point I/O. C'est notamment le cas lorsque le conteneur complet est emporté, sans retour direct dans l'espace de stockage. Dans un cycle de commande double, la machine réalise une tâche de stockage, puis va directement récupé-

rer un nouveau conteneur avant de retourner au point I/O. Le cycle de commande double est ainsi décomposable en trois segments : le déplacement de stockage, le déplacement de transition et le déplacement de récupération.

Tel que mentionné par Zollinger [15], cette automatisation présente plusieurs avantages par rapport à un système manuel. Un AS/RS nécessite moins d'espace, puisqu'il peut utiliser des racks de grande hauteur et des allées de petites dimensions. Il réduit le besoin en main-d'œuvre et les possibles erreurs de manutention. Si l'investissement initial est plus élevé, les coûts opérationnels sont plus faibles, permettant d'afficher un retour sur investissement supérieur. Une machine SR n'a pas besoin de pause, ce qui permet une production continue. L'inventaire est mieux contrôlé et les produits sont moins endommagés en réduisant la manutention. En revanche, un AS/RS présente également quelques inconvénients, tels qu'une chute de productivité lorsqu'une machine SR est à l'arrêt, un coût d'investissement important qui peut être un obstacle pour une petite entreprise, et une extension physique problématique.

Étant donné le nombre de configurations possibles pour un AS/RS, Boysen et Stephan [16] présentent une revue de littérature ayant comme objectif de classifier les AS/RSs par un 3-uplet $(\alpha|\beta|\gamma)$. α désigne la disposition générale (le type de point I/O, le nombre de machines, etc.), β représente les caractéristiques des commandes (le type de demande, l'information disponible, le rythme d'arrivée de nouvelles commandes, les échéances, les emplacements de stockage, etc.) et γ spécifie l'objectif (maximiser la cadence de production, minimiser les délais de traitement, etc.). Un des objectifs classiques est de maximiser la cadence de production qui est inversement proportionnelle au temps de déplacement des machines SR. Dans certains cas où de nouvelles commandes sont révélées régulièrement et où les échéances sont critiques, minimiser les délais de traitement peut se révéler plus pertinent.

Comme de nouvelles commandes peuvent être révélées à tout moment, une stratégie doit être déterminée pour gérer ce flux de commandes. Han et al. [17] proposent deux approches : la planification par blocs et la planification dynamique. La planification par blocs consiste à figer un sous-ensemble de commandes en fonction de leurs échéances et de planifier et traiter ces commandes avant de passer au bloc suivant. La planification dynamique implique de reconsidérer l'ensemble des commandes révélées dès qu'une nouvelle commande entre dans le système.

De Koster et al. [18] estiment que les opérations de récupération représentent environ 55% des coûts opérationnels dans un entrepôt. De ce montant, 60% est attribuable au déplacement qui est fortement dépendant de la stratégie de stockage. Ces stratégies font l'objet de nombreux travaux de recherche qui sont présentés ci-après. Même lorsque les décisions de stockage ne sont pas directement le sujet d'une étude, une hypothèse doit être faite sur l'approche qui

est mise en place. Pour cette raison, cette revue de littérature s'articule autour du type de stockage considéré dans les articles de recherche. Un type de stockage peut être une politique générale, un stockage dédié, statique ou partagé.

2.1.2 Politiques de stockage

Une politique de stockage est une règle de haut niveau qui dicte l'allocation de l'emplacement de stockage pour chaque conteneur. Ces politiques sont couramment déployées pour leur simplicité d'application et peuvent également servir de point de comparaison pour évaluer la performance d'autres approches. Roodbergen et Vis [14] décrivent quatre politiques typiquement rencontrées en pratique et dans la littérature. Ces politiques sont présentées ci-après.

— *Stockage dédié* (*dedicated storage* en anglais) : cette méthode assigne un emplacement de stockage spécifique à chaque type d'article possiblement présent dans l'entrepôt. Bien que cette méthode soit intéressante pour l'entraînement des opérateurs dans les systèmes manuels, il cause de grandes pertes d'espace et n'est pas approprié aux systèmes automatiques.

— *Stockage aléatoire* (*random storage* en anglais) : tous les conteneurs ont la même probabilité d'occuper n'importe quel emplacement de stockage. Cette méthode maximise l'utilisation de l'espace et est facile à implémenter. Cependant, il est évident qu'elle peut entraîner une mauvaise accessibilité et productivité.

— *Stockage basé sur le taux de demande* (*full-turnover-based storage* en anglais) : les conteneurs les plus souvent demandés sont assignés au plus près du point I/O. Dans certains cas, plutôt que d'utiliser directement le taux de demande, un *indice cubique par commande* (*Cube per Order Index, COI*) peut être calculé [19]. Le COI est le ratio de l'espace nécessaire à un conteneur sur le nombre de voyages nécessaires pour satisfaire la demande pendant une période donnée. Plus le COI d'un conteneur est petit, plus le conteneur est placé proche du point I/O. En pratique, des préoccupations existent autour d'un changement des taux de demande et l'arrivée de nouveaux types d'articles.

— *Stockage par classes* (*class-based storage* en anglais) : les emplacements de stockage sont regroupés en un nombre restreint de classes en fonction de leur proximité au point I/O. Ensuite, un conteneur est assigné à une classe dépendamment de son taux de demande. Cette méthode bénéficie d'une utilisation complète de l'espace et d'une bonne accessibilité mais requiert un certain nombre de décisions, comme le nombre de classes, leurs tailles et leurs positions.

Hausman et al. [20] font partie des premiers auteurs à s'être intéressés aux AS/RSs. Ils consi-

dèrent les trois dernières politiques précédemment présentées dans un système où seuls des cycles de commandes simples sont réalisés. En ordonnant les articles dans un ordre décroissant de leurs taux de demande et en discrétisant la représentation d'un rack de stockage, les auteurs calculent le temps de déplacement espéré pour la politique aléatoire et la politique basée sur le taux de demande. Comme attendu, la deuxième politique réduit d'environ 35% le temps de déplacement. Afin d'évaluer plus facilement différents types de demandes et différentes configurations des classes de stockage, ils proposent également un modèle continu du temps de déplacement au sein d'un rack. La politique basée sur le taux de demande donne les meilleurs résultats, mais la politique par classes est compétitive. Considérant les économies d'espace, la politique par classes devient un premier choix.

Dans la continuité, Graves et al. [21] incluent des cycles de commandes doubles et obtiennent des résultats comparables, tout en validant le gain de performance associé aux cycles de commandes doubles. Pour ces deux études, la représentation discrète du rack et le modèle continu présentent des résultats similaires, exception faite des distributions de la demande fortement biaisées vers quelques articles très souvent demandés.

Bozer et White [22] présentent un modèle statistique basé sur la position espérée des emplacements de stockage et de récupération lorsque la politique de stockage est aléatoire. Ils considèrent des cycles de commandes simples et doubles, ainsi que des racks de formes rectangulaires (par rapport aux racks carrés en termes de temps d'accès des études précédentes). Ils obtiennent des résultats très semblables à un modèle discret (différences inférieures à 1%) pour toutes sortes de distributions de la demande. Bien que les modèles analytiques permettent d'évaluer rapidement certains choix de configuration et le comportement général de politiques standards, ils deviennent rapidement limités pour l'étude d'autres politiques et pour le traitement de situations spéciales. Pour ces raisons, la simulation est devenue l'outil de prédilection pour évaluer la performance d'un entrepôt, même si son implémentation se révèle plus laborieuse.

Les résultats de Hausman et al. [20] et Graves et al. [21] sont validés par simulations par Schwartz et al. [23]. Leurs auteurs implémentent également une politique d'*emplacement libre le plus proche*, ou *Closest Open Location* (COL). Ils concluent que cette politique a le même niveau de performance qu'un stockage aléatoire, principalement lorsque le taux d'occupation est élevé.

Linn et Wysk [24] présentent un simulateur qui leur permet d'étudier plusieurs problèmes de décision tels des politiques de stockage, des règles d'ordonnancement ou le positionnement d'un emplacement d'attente (dwell-point positioning). Ils confirment la performance supérieure d'une politique de stockage par classes et concluent qu'il est préférable de faire attendre

la machine SR à sa dernière position visitée plutôt que de la déplacer par anticipation. Les mêmes auteurs [25] incluent des demandes saisonnières, considèrent les temps d'accélération et de décélération de la machine SR et appliquent une règle d'ordonnancement qui minimise les temps de traitement. Ils suggèrent la réévaluation périodique de l'affectation des articles aux classes, ainsi que la dimension de ces classes pour une adaptation à un changement de demande. Ces sujets sont traités par Van den Berg [26] qui propose un algorithme de programmation dynamique pour affecter les articles aux classes de manière optimale. Cet algorithme minimise les temps de déplacement pour des cycles de commandes simples. L'auteur optimise également l'espace associé à chaque classe pour contrôler les risques de débordement du stock.

Van den Berg et Gademann [27] présentent une étude de simulation qui couvre une grande variété de règles de décision et évaluent leurs performances indépendamment ou en corrélation avec les autres (en combinant une règle de stockage avec une règle d'ordonnancement). Parmi leurs conclusions, ils déterminent qu'avec des cycles de commandes doubles, sélectionner l'emplacement de stockage le plus proche du prochain emplacement de récupération donne le meilleur temps de traitement moyen. Ils trouvent également qu'ordonner les commandes permet de réduire le temps de traitement moyen mais est sujet à d'importants retards pour certaines commandes. Pour cette raison, ils suggèrent de traiter les commandes dynamiquement, par ordre d'échéance.

Gagliardi et al. [28, 29] présentent en détail un modèle de simulation à événements discrets particulièrement réaliste ayant pour vocation d'être applicable à de nombreuses configurations. Ce simulateur est utilisé pour comparer la politique de stockage aléatoire avec celle basée sur le taux de demande et la politique par classes. Il apparait que lorsque certaines conditions réalistes sont reproduites, la politique basée sur le taux de demande peut se révéler nettement moins efficace que les autres politiques. Les auteurs soulignent l'importance d'une étude poussée par simulations pour chaque type d'application, car les résultats théoriques peuvent fortement différer de la réalité.

2.1.3 Stockage dédié

Dans cette section, l'emplacement des conteneurs est connu et figé à l'avance. Plusieurs travaux de recherche font cette hypothèse et considèrent le problème d'ordonnancement des commandes.

Lee et Schaefer [30] proposent une méthode exacte simple pour créer des cycles de commandes doubles. Ils résolvent un problème d'affectation afin de coupler une tâche de stockage avec une tâche de récupération. Ils proposent également deux environnements pour gérer le flux

entrant de commandes : la planification par bloc [17] et la planification sur fenêtre filante. De plus, ils présentent une méthode heuristique qui sélectionne la paire de tâches de stockage et de récupération la plus proche. De manière similaire, Mahajan et al. [31] présentent également une heuristique de plus proche voisin et évaluent sa performance avec un modèle analytique et une étude de simulation. Ils trouvent que leur méthode améliore de 5 à 15% la productivité par rapport à une règle de premier arrivé, premier servi (First-Come First-Served, FCFS).

Selon Van den Berg et Gademann [32], le problème d'ordonnancement des commandes est NP-difficile en général. Cependant, dans le cas spécial du stockage dédié, les auteurs présentent un modèle de transport qui se résout en temps polynomial. Les tâches de stockage sont traitées en ordre FCFS et les points d'entrées et de sorties peuvent être à des emplacements distincts. De plus, la machine SR peut réaliser des cycles de commandes simples ou doubles. Puisque les déplacements chargés sont fixes (du fait du stockage dédié), l'objectif est de déterminer les déplacements faits à vide. Un graphe biparti sépare les tâches de stockage et de récupération et une solution optimale au problème d'affectation correspond à une solution optimale du problème d'ordonnancement. Les auteurs présentent également deux méthodes heuristiques qui peuvent s'avérer utiles dans un environnement dynamique.

Dans un environnement différent, Alonso-Ayuso et al. [33] considèrent un entrepôt avec des chariots élévateurs et plusieurs allées. Des camions de livraison se présentent pour charger un certain nombre d'articles de références données. Ces références peuvent être trouvées à différents emplacements qui correspondent à différents temps d'accès. Une méthode à deux phases est proposée pour optimiser la récupération des palettes. La première phase consiste à sélectionner les palettes qui serviront à satisfaire la demande. Ceci est réalisé avec un MIP qui minimise le temps d'accès. La seconde phase définit l'ordonnancement des palettes sélectionnées pour la cueillette. Les auteurs présentent une heuristique en deux phases qui consiste en une étape de construction puis une étape d'amélioration. Cependant, il semble que les chariots réalisent des cycles de commandes simples, ce qui est surprenant d'un point de vue pratique. Aussi, les temps de disponibilité des chariots et les temps de livraison associés à l'arrivée des camions ne sont pas considérés.

Comme mentionné en Section 2.1.2, le stockage dédié cause des pertes d'espace et ne tire pas profit de la flexibilité de stockage d'un système automatisé. Puisqu'un conteneur peut être dynamiquement affecté à n'importe quel emplacement de stockage libre, des opportunités en termes de longueurs de cycles et d'utilisation de l'espace peuvent être utilisées. Certaines des politiques générales de stockage présentées précédemment utilisent déjà cette affectation dynamique. D'autres approches ont été proposées dans la littérature et peuvent être séparées en deux catégories : un *stockage statique* ou un *stockage partagé*. Un stockage statique suppose

que les conteneurs nouvellement affectés à un emplacement ne seront pas de nouveau récupérés avant le prochain horizon. Typiquement, seuls les emplacements déjà disponibles au début de l'horizon sont considérés comme emplacements de stockage potentiels. Un stockage partagé relaxe ces hypothèses. Un conteneur affecté à un emplacement peut être récupéré dans le même horizon de temps et n'importe quel emplacement disponible à un instant donné peut être utilisé.

2.1.4 Stockage statique

Yin et Rau [34] traitent les tâches de stockage dans un ordre FCFS et considèrent trois règles pour créer des cycles de commandes doubles :

— FCFS/STT : FCFS pour l'ordre des requêtes de récupération et temps total le plus court (ou Shortest Total Time, STT) pour sélectionner l'emplacement de stockage et le conteneur.

— STT : temps total le plus court entre tous les emplacements de stockage et des conteneurs candidats à une récupération.

— SDT/STT : échéance la plus courte (Shortest Due Time, SDT) pour l'ordre des requêtes de récupération et STT pour sélectionner l'emplacement de stockage et le conteneur.

Les auteurs proposent également un algorithme génétique pour déterminer quelle règle utiliser à quel moment. Avec une étude de simulation, ils obtiennent de meilleurs résultats lorsque la même règle est utilisée en permanence.

Dans un contexte similaire, Hachemi et al. [35] traitent les requêtes de stockage en ordre FCFS et ordonnent les requêtes de récupération pour minimiser le temps de déplacement. Il y a également plusieurs emplacements de stockage disponibles et plusieurs conteneurs pour chaque type d'article. Ils résolvent le problème itérativement (après chaque cycle) avec un MIP dont l'objectif est de minimiser les STTs.

Gagliardi et al. [36] proposent un nouvel environnement de gestion du flux de commandes qui se positionne entre la planification par blocs et la planification dynamique. Cet environnement requiert la définition de deux paramètres : l'horizon de planification h et l'horizon fixe f qui détermine le nombre de cycles entre chaque replanification. Les auteurs affirment que le choix des valeurs de ces paramètres est essentiel à un bon rapport entre la qualité de la solution et la complexité du problème. De nouveau, les tâches de stockage sont traitées en ordre FCFS, puis les auteurs adaptent des règles de décision à leur environnement : plus proche voisin (ou Nearest Neighbour, NN), temps d'accès et de transition le plus court (ou Shortest Leg, SL) et STT. La contribution principale de cet article est la présentation

d'un modèle d'ordonnancement MIP. Les résultats sont prometteurs mais les auteurs reconnaissent également les limitations de leur modèle statique qui n'utilise que les emplacements initialement disponibles. Ils se restreignent au cas statique à cause de la complexité qui est déjà associée à leur modèle. En effet, ce modèle utilise une indexation quadruple pour ses variables binaires, pour lesquelles une variable correspond à une combinaison d'une tâche de stockage avec une tâche de récupération, un emplacement de stockage et un emplacement de récupération.

Dans un type d'application différent, Wauters et al. [37] étudient un système à charge minimale (miniload) avec une machine SR à double capacité. Les auteurs présentent dans un premier temps une décomposition en deux sous-problèmes : l'allocation de stockage et l'ordonnancement des requêtes. L'allocation de stockage est faite à partir de deux stratégies (séquentielle et simultanée) qui utilisent deux critères C1 (niveau de priorité d'une requête) et C2 (variante du temps d'accès d'un emplacement). Par la suite, les auteurs proposent un modèle d'ordonnancement MIP ainsi qu'une stratégie de branchement. Le modèle est capable de considérer l'allocation de stockage mais à un coût computationnel largement supérieur. Les instances de plus grandes dimensions sont résolues avec une méthode métaheuristique (*Step Counting-Hill Climbing*).

Roozbeh Nia et al. [38] considèrent un système à plusieurs racks où un article peut être récupéré dans plusieurs conteneurs. Leur objectif est spécifiquement de minimiser les émissions de gaz à effet de serre. Ils développent un algorithme de colonie de fourmis ainsi qu'un algorithme génétique pour optimiser leur modèle dans un environnement dynamique où, à chaque fois qu'une nouvelle commande est révélée, les k commandes les plus urgentes sont réoptimisées.

2.1.5 Stockage partagé

Lee et Schaefer [39] présentent une méthode quasi-optimale pour l'ordonnancement des tâches de récupération et l'allocation de stockage lorsqu'une commande est associée à exactement un conteneur. Ils comparent leur méthode avec les règles NN, SL et STT. L'idée générale est de formuler le problème comme un problème d'affectation. Une telle affectation peut s'avérer irréalisable si un tour est créé (affectation d'une tâche de stockage à un emplacement occupé), mais elle permet d'obtenir une borne inférieure. Cette solution est ensuite corrigée avec des heuristiques telles que SL et STT afin d'obtenir une borne supérieure réalisable. Par la suite, en utilisant l'algorithme de Murty pour les problèmes d'affectation, ils considèrent la seconde meilleure affectation et répètent le processus. Ils obtiennent de très bons résultats et observent que leur méthode converge plus rapidement lorsqu'il y a plus d'emplacements

libres initialement (ce qui crée moins de tours). En revanche, bien qu'un emplacement libéré dans l'horizon considéré puisse être utilisé pour un stockage, il est supposé qu'un conteneur récemment entreposé ne sera pas de nouveau récupéré.

Le terme de *stockage partagé* est mentionné explicitement pour la première fois par Goetschalckx et Ratliff [40]. Dans cet article, seuls des cycles de commandes simples sont considérés. Dans le cas d'un système parfaitement équilibré (même nombre d'arrivées et de départs de conteneurs à durées de séjour égales), l'espace et de temps de déplacement peuvent être optimisés de manière optimale avec des règles simples. Il suffit d'allouer tous les conteneurs d'une durée de séjour p à une zone z_p dont la distance au point I/O dépend de p. Lorsque le système n'est pas équilibré, une approche similaire peut être dérivée sous forme d'une méthode heuristique qui donne de meilleurs résultats que le stockage dédié.

Montulet et al. [41] cherchent à minimiser le temps d'accès avec des cycles de commandes simples et en supposant connus les temps d'arrivée et de départ des conteneurs. Les auteurs proposent une méthode exacte basée sur la génération de colonnes dans laquelle une variable correspond à des conteneurs compatibles pour occuper le même emplacement. L'allocation de stockage est faite par classes plutôt que par emplacements exacts. Ils obtiennent de bons résultats pour des petites instances. Pour des instances de plus grandes dimensions, ils proposent une heuristique qui consiste à chercher un plus long chemin dans un graphe où les nœuds correspondent à des temps d'arrivées et de départs et les arcs à un séjour d'un conteneur. Les arcs ont un coût $K + DS$, où K est une constante plus élevée que la durée totale de l'horizon et DS est la durée de séjour. Cette heuristique donne des résultats comparables à la méthode exacte. Il est important de noter qu'en minimisant le temps d'accès et en supposant connus les temps d'arrivée et de départ, ces méthodes ne garantissent pas que la machine SR puisse satisfaire la demande en pratique.

Chen et al. [42] considèrent un problème similaire mais avec des cycles de commandes doubles. Les auteurs formulent le problème avec un MIP qu'ils sont en mesure de résoudre avec Cplex pour de très petites instances. Pour de plus grandes instances, une méthode à deux phases est proposée. La première phase alloue les emplacements de stockage avec la méthode de Montulet et al. [41]. La seconde phase minimise les déplacements de transition avec un problème d'affectation. De plus, le résultat final est encore amélioré avec l'application d'une méthode Tabou. De nouveau, une telle solution ne garantit pas d'être réalisable en pratique.

Dans une application différente, Yang et al. [43] s'intéressent au stockage partagé pour un système avec une machine SR à charges multiples. Dans ce problème, les emplacements de récupération sont connus. L'objectif est de déterminer le premier emplacement de stockage parmi les emplacements disponibles, puis de déterminer la séquence de récupérations/stockages.

Notons que si le premier emplacement était connu, le problème reviendrait à une tournée de véhicules avec des contraintes de cueillettes et de livraisons. Un programme en nombres entiers est résolu par Cplex pour de petites instances et une recherche à voisinage variable est utilisée pour les instances plus grandes. Un point essentiel est qu'un tel système est capable de stocker et récupérer un conteneur au même emplacement, ce qui n'est pas le cas d'un système à charge simple.

2.2 Centres de distribution robotisés

Depuis quelques années et suivant le rapide engouement pour les RMFSs, une littérature spécifique s'est développée. La différence principale entre un RMFS et un AS/RS traditionnel est le grand nombre de robots (pouvant être vus comme des machines SR) qui opèrent simultanément dans l'ensemble de l'espace de stockage. Cela génère de nouveaux défis logistiques en termes de complexité du système et de nouvelles opportunités de prises de décisions.

2.2.1 Modèles analytiques

Plusieurs articles proposent des modèles analytiques afin d'être en mesure d'évaluer rapidement des indicateurs de performance. Lamballais et al. [44] proposent un modèle de files d'attente semi-ouvert pour estimer le flux de production maximal, le temps de cycle moyen par commande et le niveau d'utilisation des robots, en fonction de la configuration physique de l'entrepôt et de stratégies de zonage des robots. Ils démontrent la précision de leur modèle comparativement à des études de simulations et ils observent que le flux de production est plus influencé par l'emplacement des stations de cueillette que par la dimension de l'espace de stockage.

Zou et al. [45] présentent également un modèle de files d'attente semi-ouvert mais pour évaluer des règles d'affectation des robots aux stations de cueillette. Ils démontrent la supériorité de leur règle par rapport à une affectation aléatoire.

Yuan et al. [46] proposent un modèle fluide pour estimer la performance de politiques de stockage *basées sur la vélocité* (ou *velocity-based*) en termes de temps de stockage et de récupération. Ces politiques basées sur la vélocité sont une adaptation des politiques de stockage par classes à un système à plusieurs stations de cueillette. Ils trouvent que lorsque les articles sont affectés aléatoirement dans les étagères, une politique à deux ou trois classes réduit le temps de déplacement chargé entre 6 et 12%. De plus, si les articles sont affectés aux étagères relativement à leurs taux de demande, ce temps de déplacement peut être réduit jusqu'à 40%.

En utilisant un modèle de files d'attente fermé, Roy et al. [47] évaluent l'impact d'allouer les robots exclusivement à des tâches de récupération, à des tâches de réapprovisionnement ou à une combinaison des deux. Ils trouvent qu'une combinaison des deux types de tâches réduit le temps de cueillette par 30% mais augmente le temps de réapprovisionnement par un facteur trois. Ils étudient également l'allocation de stockage en classes et ils concluent qu'affecter un robot à la classe la moins congestionnée résulte en une performance équivalente à une politique par classes basée sur le taux de demande.

Lamballais et al. [48] proposent un modèle de files d'attente semi-ouvert pour déterminer le nombre d'étagères par types d'articles, le ratio du nombre de stations de cueillette par rapport au nombre de stations de réapprovisionnement et le niveau d'inventaire par étagère. Ils concluent qu'un gain significatif de performance est obtenu lorsqu'un type d'article est affecté à de multiples étagères, si le ratio de stations de cueillette et de réapprovisionnement est déterminé avec soin en fonction de l'application et si une étagère est réapprovisionnée avant d'être vide.

2.2.2 Ordonnancement des commandes

D'autres articles de recherche se concentrent sur le traitement des commandes aux stations de cueillette ou par un robot spécifique. Boysen et al. [49] introduisent un MIP afin de minimiser le nombre de visites d'étagères à une station de cueillette en synchronisant l'ordonnancement des commandes et l'arrivée des étagères à la station. Pour résoudre le modèle, ils proposent différentes méthodes heuristiques basées sur la programmation dynamique et la métaheuristique de recuit simulé. Par simulations, ils montrent que leur méthode peut réduire de moitié la taille de la flotte de robots dans le meilleur cas.

Gharehgozli et Zaerpour [50] s'intéressent au problème d'ordonner un bloc de commandes à un robot unique afin de minimiser le temps de déplacement de ce robot. Ils considèrent deux cas où une étagère doit être entreposée au même emplacement que sa position initiale, ou si elle peut être entreposée dans un ensemble d'emplacements déterminés. Ils démontrent l'efficacité de leur résolution avec une méthode de recherche adaptative de voisinage large (ou adaptive large neighbourhood search) et réduisent de 24% le temps de déplacement par rapport à un ordonnancement aléatoire.

Valle et Beasley [9] présentent un modèle en nombres entiers et deux méthodes matheuristiques pour simultanément allouer un bloc de commandes et des étagères aux stations de cueillettes afin de minimiser le nombre de visites des étagères. De plus, ils formulent un deuxième modèle afin de déterminer une séquence réalisable des visites des étagères à une station.

Bolu et Korcak [51] proposent une approche de planification de tâches pour minimiser le nombre d'étagères nécessaires pour satisfaire un bloc de commandes. Après que les étagères aient été sélectionnées, ils adaptent le système dynamiquement afin d'allouer les tâches de récupération aux robots et aux stations. Par simulations, ils démontrent la capacité de leur méthode à réduire de manière significative le temps de traitement ainsi qu'à maximiser le nombre d'articles récupérés par visite d'une étagère.

2.2.3 Allocation d'emplacements de stockage

Tout comme pour les AS/RS, le problème d'allocation d'emplacements de stockage est sujet de plusieurs travaux de recherche. Krenzler et al. [52] présentent un modèle MIP ayant comme objectif de minimiser le temps de déplacement chargé (trajet de stockage et trajet de récupération) pour un bloc de commandes. Ils considèrent connus les temps de visites des étagères aux stations de cueillette et que les files d'attente aux stations sont toujours pleines, ce qui a pour effet de libérer plus d'emplacement dans l'espace de stockage. Les auteurs présentent plusieurs méthodes de résolution et démontre la supériorité de leur approche par rapport à un stockage aléatoire.

Weidinger et al. [53] présentent également un modèle MIP pour minimiser le temps de déplacement chargé sous la forme d'un problème de planification d'intervalles. Comme précédemment, ils supposent connus les temps de séjour à la station de cueillette. En tant que problème d'intervalles, deux étagères peuvent occuper le même emplacement de stockage tant que leurs intervalles de stockage ne se chevauchent pas. Ils proposent une méthode de résolution par une méthode matheuristique de recherche adaptative large et obtiennent de meilleurs résultats en termes de temps de déplacement chargé par rapport aux politiques de stockage de référence. D'un point de vue pratique, ils notent l'excellente performance de la politique Shortest Leg (SL) qui n'augmente le temps de déplacement que de 3.49% en opérant parfaitement en temps réel et sans nécessiter de regroupements de commandes.

Mirzaei et al. [54] traitent le problème d'une autre perspective. Ils considèrent l'affinité des articles (probabilité que deux articles soient commandés ensemble), ainsi que le taux de demande pour affecter conjointement les articles aux étagères et les étagères aux emplacements de stockage pour minimiser le temps de récupération avec un modèle MIP. Leur modèle permet de réduire le temps de récupération par 40% comparativement à une politique de stockage basée exclusivement sur le taux de demande ou par classes.

Li et al. [55] considèrent le problème d'affectation d'articles aux étagères et de l'allocation des étagères aux emplacements de stockage. Leur objectif est de réduire la consommation d'énergie des robots qui dépend principalement de la distance parcourue par les robots et

le nombre d'arrêts/départs qu'ils réalisent. Dans un premier temps, l'affectation des articles aux étagères est réalisée avec un programme en nombres entiers qui minimise la somme des coefficients de similarité entre articles au sein d'une même étagère. Ensuite, un autre modèle est proposé pour allouer les emplacements de stockage en équilibrant les temps d'accès et les distances entre étagères qui risquent d'être requises en même temps et ainsi de causer des arrêts des robots. Leurs résultats montrent que leur approche peut réduire de 22% la consommation d'énergie.

2.2.4 Autres thématiques

D'autres articles traitent des problèmes de décisions connexes aux sujets précédents. Zhang et al. [56] définissent le problème d'allocation de robots aux tâches de stockage suivant leur ordonnancement aux stations de cueillette. Ils formulent le problème comme un problème d'ordonnancement sous contraintes de ressources et proposent de le résoudre avec une génération séquentielle de l'ordonnancement et un algorithme génétique. Leur méthode génère de meilleurs résultats que des règles générales d'ordonnancement.

Xie et al. [57] proposent un modèle MIP pour affecter simultanément les étagères requises aux stations et les commandes aux stations afin de minimiser le nombre de visites des étagères. Ils considèrent également la division des commandes contenant plusieurs articles pour permettre leur récupération à différentes stations de cueillette. Ils utilisent le simulateur à événements discrets RAWSim-O [58] qui reproduit les opérations d'un RMFS de manière particulièrement réaliste. Avec la division des commandes, leur méthode parvient à augmenter la productivité du système de 46%.

Avec ce même simulateur, Merschformann et al. [13] évaluent l'impact sur la productivité d'un grand nombre de règles de décision. Ces règles traitent des problèmes de décision allant de l'affectation de stations de cueillette et de réapprovisionnement, à la sélection d'étagères, ou encore à l'allocation d'emplacements de stockage. Les diverses règles pour différents problèmes sont combinées afin d'être évaluées conjointement. Ces combinaisons génèrent des résultats d'une grande variabilité, ce qui fait conclure aux auteurs qu'une sélection minutieuse devrait être faite pour chaque application. Leurs résultats mettent également en évidence l'impact important de la règle d'affectation d'une station de cueillette. Les auteurs notent aussi la forte interdépendance entre les règles, ce qui suggère le bénéfice probable d'approches intégrées.

CHAPITRE 3 DÉMARCHE ET ORGANISATION DE LA THÈSE

Cette thèse est construite autour de trois articles de recherche. Ces articles font partie d'une démarche de réflexion commune et sont ainsi fortement reliés. Initialement, ce projet de recherche a débuté avec un stage Mitacs à JDA Software à Montréal (aujourd'hui Blue Yonder), entreprise spécialiste en gestion de la chaîne d'approvisionnement. Le partenaire industriel était intéressé par un nouveau type d'entrepôt spécialisé dans le commerce en ligne, ainsi que par la prise de décision en temps réel (des approches d'optimisation stochastique en ligne). Par la suite, un second stage de recherche a été réalisé à Element AI, spécialiste en machine learning, qui a depuis été incorporé ServiceNow. Ce stage a orienté la recherche vers des approches d'apprentissage automatique, qui ont notamment favorisé le développement de politiques rapidement déployables en temps réel.

Le chapitre 4 présente le premier article, intitulé en anglais *Robotic Mobile Fulfillment Systems : a mathematical modelling framework for e-commerce applications*. Cet article a été soumis en octobre 2020 à *International Journal of Production Research* et publié le 19 mai 2021. Ce premier article a été motivé par les premières avancées liées au deuxième article. En effet, il était initialement envisagé de développer des méthodes d'optimisation stochastique en ligne pour le problème intégré d'allocation d'emplacements de stockage et d'ordonnancement des commandes dans un RMFS. Cependant, il est rapidement devenu clair que la complexité du système et la notion de prise de décision en temps réel rendaient la définition même d'un problème difficile. Outre la stochasticité liée à l'arrivée continue de nouvelle information, la nature dynamique des opérations rendait impossible la formulation du problème sous la forme d'un programme mathématique classique. De plus, une multitude d'hypothèses devaient être faites pour traiter les différents cas particuliers et autres problèmes opérationnels non considérés. Nous avons alors pensé à proposer un modèle de programmation dynamique stochastique (qui peut paraître un peu procédural) afin de formaliser le cadre de prise de décision en temps réel. De plus, compte tenu du vif et récent intérêt pour ce type d'entrepôt dans la littérature, le modèle proposé considère une large gamme de problèmes opérationnels, dont la plupart ne sont pas traités dans cette thèse, afin de sensibiliser la recherche vers la prise de décision en temps réel. Enfin, le modèle a été illustré sur le sous-problème d'allocation d'emplacements de stockage en comparant des politiques communément rencontrées dans la littérature. Ces résultats servent de points de référence pour les méthodes développées dans le reste de la thèse.

Le chapitre 5 correspond au deuxième article, intitulé *E-commerce warehousing : learning a*

storage policy, qui a été soumis à *European Journal of Operational Research* en janvier 2021. À l'heure actuelle, cet article n'a pas encore reçu de retour de la part du journal. Le sujet de cet article est l'apprentissage d'une politique de stockage par renforcement pour minimiser le temps de déplacement moyen des robots. Inspirée par la politique de stockage par classes, l'approche proposée cherche à allouer une étagère à une zone (au sein de laquelle le stockage sera réalisé arbitrairement) plutôt qu'à un emplacement exact. À la suite de l'étude des politiques de référence, il apparait qu'un compromis existe entre les effets à long terme liés à une demande biaisée (motivation première d'une politique par classes) et les opportunités à court terme visant à réaliser des cycles les plus courts possibles (politique gloutonne). L'apprentissage par renforcement se voit alors proposée une représentation partielle de l'état du système qui permette de tirer profit de ce compromis. Cette représentation contient de l'information sur le taux de demande de l'étagère à entreposer (effet à long terme) et l'emplacement de la prochaine étagère à récupérer (effet à court terme), ainsi que d'autres informations relatives au niveau d'occupation des zones de stockage. Les résultats obtenus sont supérieurs aux meilleurs résultats des politiques de référence, présentant un gain de performance allant de 3.21 à 4.83%. Par la suite, une stratégie d'exploration de trajectoires est proposée afin d'améliorer la performance de la politique apprise mais également celle des autres politiques. Cette exploration vise à utiliser plus d'information sur les commandes déjà révélées à un instant donné. Les résultats sont alors sensiblement améliorés avec, par exemple, un gain additionnel de 2.78 à 4.54% pour la politique apprise.

Enfin, le chapitre 6 présente le troisième article, intitulé *Supervised learning and tree search for real-time storage allocation in Robotic Mobile Fulfillment Systems*. Cet article a récemment été soumis à *INFORMS Journal on Optimization* en mai 2021. La motivation de cet article est venue du gain de performance convaincant obtenu par une stratégie d'exploration de trajectoires. L'objectif de cet article est alors d'apprendre une politique de stockage qui sélectionne cette fois des emplacements exacts et qui utilise des techniques d'exploration d'arbres de recherche en phase d'apprentissage et/ou de déploiement. Un élément essentiel aux méthodes d'explorations proposées dans ces travaux de recherche est la capacité du simulateur à copier son état courant afin de simuler un futur alternatif du système avant que la prise de décision finale ne soit faite. Un apprentissage par renforcement n'est alors pas la seule approche candidate puisqu'il s'agit en pratique de réaliser un grand nombre d'essais-erreurs entièrement dépendants de l'évolution de l'environnement du système. Une alternative consiste à utiliser des méthodes d'exploration d'arbres de décision telle que la recherche arborescente Monte-Carlo à chaque prise de décision. La contrepartie est que chacune de ces recherches prend un temps de calcul important, ce qui rend l'approche difficilement déployable. Cependant, cette recherche peut facilement être faite hors-ligne en parallèle afin

de rapidement accumuler une vaste expérience d'excellente qualité. Cette expérience peut, par la suite, être apprise de manière supervisée avec un choix de représentation adéquat (ici en utilisant les coordonnées spatiales des emplacements de stockage). Finalement, la politique ainsi apprise peut être directement déployée en temps réel ou peut être utilisée pour guider efficacement une nouvelle exploration d'arbre. Les résultats obtenus améliorent ainsi la meilleure politique de référence jusqu'à 13.7%.

CHAPITRE 4 ARTICLE 1 : ROBOTIC MOBILE FULFILLMENT SYSTEMS : A MATHEMATICAL MODELLING FRAMEWORK FOR E-COMMERCE APPLICATIONS

Adrien Rimélé[ab], Michel Gamache[ac], Michel Gendreau[ab], Philippe Grangier[d], Louis-Martin Rousseau[ab]

[a]Department of Mathematics and Industrial Engineering, Polytechnique Montréal, [b]CIRRELT, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, [c]GERAD, Group for Research in Decision Analysis, [d]IVADO Labs

Cet article a été soumis à *International Journal of Production Research* le 20 octobre 2020 et a été publié le 19 mai 2021.

ABSTRACT

Robotic Mobile Fulfillment Systems (RMFSs) are a recent type of automated warehouse deployed in e-commerce. In this parts-to-picker system, a fleet of small robots is tasked with retrieving and storing shelves of items in the warehouse. Due to the nature of the e-commerce market, and the high flexibility of RMFSs, there are many opportunities to improve the productivity of the warehouse by optimising operational decisions. Online retailers promise extremely fast deliveries, which requires that new orders be included in the set of requests to fulfil as soon as they are revealed. For this reason, and because of the very dynamic nature of the robots' cycles, decision-making needs to be done in real time, in an uncertain environment. Because such a problem often lacks a formal description, we propose a mathematical framework that models the operational decisions taking place in an RMFS as a stochastic dynamic program. Our objective is to formalise optimisation opportunities, to allow researchers to develop more advanced methods in a well-defined environment. Embedded in a discrete event simulator, this model is illustrated by simulations to compare against standard storage decision rules.

KEYWORDS

warehouse ; e-commerce ; Robotic Mobile Fulfillment System ; stochastic dynamic programming ; mathematical modelling ; simulations

4.1 Introduction

Warehouses play a central role in any supply chain. Regarding the impact of warehousing activities on the economy, Tompkins and Smith [59] suggest that the real value of warehousing relies on having the right product in the right place at the right time. According to Gu et al. [60], a warehouse's primary purposes are to act as a buffer to adapt to the variability of production flow ; to consolidate products which come from different sources and must be grouped for shipping to customers ; and to add marginal value to the product such as pricing, labelling or customisation. Gu et al. [60] divide the operations that take place in a warehouse into four types : receiving, storing, picking, and shipping. Picking operations are of critical importance, as emphasised by de Koster et al. [18] and Shah and Khanzode [61], who estimate that these operations constitute roughly 55% of all warehouse operating expenses, 60% of which is attributable to travelling within the warehouse.

In the Business-to-Consumer (B2C) segment, the growth of e-commerce in recent years is revolutionising the full sector. In 2016 alone, e-commerce sales increased by 23%, representing 8.7% of the total retail market [2]. More than the growth of the e-commerce market, its specificities are the biggest challenge to warehousing operations. E-commerce involves enormous volumes of very small orders (1.6 lines per order on average) ; requires an enormous assortment of items ; and faces uncertain demand with fast-changing trends, promotional events, etc. Facing ample competition, online retailers cannot rely on customer loyalty ; instead, they operate under continuous pressure and need to fulfil orders as quickly as possible [5], compelling them to offer differentiated services such as same-day delivery from Amazon.

To deliver to customers ever faster, and because 'adaptation is urgent', new types of automated warehouses have appeared [62]. One of them is the Robotic Mobile Fulfillment System, a parts-to-picker system where a fleet of small robots move between the storage area, where they retrieve and store full shelves of items, and picking stations, where human operators pick the required items from the shelves. Such a system is particularly well-suited for e-commerce. First, it presents great real-time decision-making opportunities, mainly because of its storage flexibility and its fleet of robots operating simultaneously. Indeed, when returning a shelf of items to the storage area, a robot can dynamically choose a new storage location to improve the system's general performance : this makes the full layout adaptable. Also, several other decisions can be made online to improve the overall performance, such as the scheduling of the incoming orders, the selection of a shelf, or which picking station to use. This allows adapting quickly to a fast-changing trend. Then, the system is easily expandable when the demand is higher, and because of the enormous number of small orders, numerous robots can simultaneously retrieve items from the whole storage area (which is, for instance, hardly

done with a traditional crane system).

The contribution and foremost objective of this paper is to present a mathematical framework to model the dynamic decision-making occurring during the storage and retrieval operations in an RMFS. The nature of e-commerce orders, as well as uncertain processing times and demands, justify online decision-making, without batching. Because of the dynamic aspect and complexity of the operational decision-making, most of the current practice and research relies on a combination of high-level decision rules evaluated through simulations or analytical models. Often, such real-time problems lack a rigorous mathematical framework. We believe that formalising the decision process with a stochastic dynamic model facilitates the development of more advanced solution approaches.

This work is presented as follows. Section 4.2 describes the Robotic Mobile Fulfillment System, as well as its optimisation opportunities, while Section 4.3 presents a literature review of existing work on this storage system. Section 4.4 presents the mathematical model, which formalises the problem under Powell's unified framework of stochastic optimisation [63]. Section 4.5 demonstrates a *basic* usage of the framework by adapting typical decision rules in a simulation study. Finally, Section 4.6 offers some conclusions and explores future research opportunities.

4.2 Robotic Mobile Fulfillment Systems

The Robotic Mobile Fulfillment System, also called *rack-moving mobile robot warehouse* or *Kiva warehouse*, was first introduced by Kiva Systems in 2006 (renamed Amazon Robotics after acquired by Amazon in 2012) [4, 6, 7]. Similar systems using small robots to lift shelves of items have since been developed by other companies, including Alibaba, Knapp, Swisslog, Locus Robotics, and others [4, 8, 64].

As mentioned in the introduction, an RMFS is a parts-to-picker system in which a fleet of small robots (sometimes called AGVs for automated guided vehicles) is in charge of retrieving and storing shelves (also called pods or bins) of items in the storage area. These robots move around following a system of waypoints organised as a regular grid. Human operators stand at picking stations, ready to pick items from a shelf to send them to the downstream system. The life cycle of a robot corresponds to a *dual command cycle*, which defines an immediate succession of a storage task and a retrieval task. When a robot leaves a picking station, any open storage location can be selected. Once the shelf has been stored, the robot goes directly to the next retrieval location to load another shelf and bring it to a picking station where it may wait in line behind other robots while a human operator is picking items. Interestingly,

when a robot is loaded, it moves along the aisles, but when it is unloaded, it can pass under the stored shelves to avoid conflicts.

A typical and more common Automated Storage and Retrieval Systems (AS/RS) corresponds to a single unit-load (or mini-load if the load is a tote bin, possibly containing multiple item types) aisle-captive system in which there is one crane per aisle that moves both horizontally and vertically [14,61]. RMFSs have several advantages over such AS/RSs [6]. In particular, the operators located at picking stations can receive shelves from all of (or most of) the storage area, so fewer operators are needed. There is better accountability and accuracy because orders are generally fully processed by one operator, which also reduces delays as it eliminates downstream dependencies. Since any location is accessible by every robot, and there is a large fleet of robots, there is no single point of failure. The system is easily implementable in any environment, and it is easy to extend if needed. Finally, and this is of great interest in terms of optimisation : RMFSs offer enormous storage flexibility. The allocation of a shelf to a storage location can be modified after every picking operation, which allows for the layout to be adapted in real time, whether in response to changing demand or simply to leverage immediate opportunities. For a more detailed and technical review on RMFSs, we refer the reader to the following papers : [6, 7, 10, 11].

The logic employed by traditional warehouse management systems fails to accommodate the extremely fast delivery times necessary in the e-commerce sector. E-commerce orders must be fulfilled as quickly as possible, necessitating what Banker [12] calls *order streaming logic*, which 'drops these orders [...] to the floor as soon as they are received'. This is of importance in optimisation because it limits the possibility of processing orders in batches, but instead favours dynamic decision-making in which new orders can be revealed at any time.

4.3 Literature review

The literature on RMFSs includes research on AS/RSs, automated warehouses that were introduced in the 1970s (see [14] for a survey of literature on the topic). While there are key differences between RMFSs and common AS/RSs, some features are similar. Dual command cycles and the question of block (batching) sequencing versus dynamic sequencing are already central topics in AS/RS research. In the case of dynamic sequencing, most if not all works rely on decision rules for storage allocation and sequencing of orders [27,28]. Common storage allocation rules include : *random storage* ; *turnover-based storage* (the higher the turnover of a bin, the closer to the operator) ; *closest open location* ; and *shortest leg* (select the open storage location closest to the next retrieval location). Through simulations, van den Berg and Gademann [27] conclude that for dual command cycles, the shortest leg rule performs

the best. Gagliardi et al. [36] present a discrete event simulator and compare a random, a full turnover-based and a class-based storage policy (a bin is assigned to a class based on its turnover rate ; its location within the class is random). Their results show that when realistic conditions are met, full turnover-based storage performs much worse than the other two policies. Their takeaway is that for real-world applications, theoretical results (such as analytical models) may be far from reality. Careful simulation studies should be performed for every special case since it is unlikely that a single simple rule could perform well in all conditions.

These earlier studies have greatly influenced more recent works on RMFSs. An RMFS could even be seen as a special (and more complex) type of AS/RS, designated as a mini-load with dual command cycles system with a fleet of non-aisle-captive robots. Several works such as [44] and [48], focus on developing queueing networks to analytically estimate KPIs, such as maximum order throughput or average cycle time. Those models facilitate quick estimates of the impact of strategic decisions such as layout design, the number of shelves used to store an item, the ratio of the number of replenishment stations to picking stations, etc. Zou et al. [45] use a semi-open queueing network to study assignment rules of robots to picking stations and demonstrate that their proposed rule outperforms a random assignment.

Boysen et al. [49] study the problem of sequencing the processing of a set of orders at a picking station, to minimise the number of shelves that need to be brought to fulfil these orders. With diverse methods, such as a mixed integer programming (MIP) model, a decomposition method, and heuristics, they manage to reduce the fleet of robots needed by half. Bozer and Aldarondo [65] compare a traditional mini-load AS/RS with an RMFS, to identify configurations that yield a similar throughput. For a fixed set of parameters and defined decision rules, they conclude that a mini-load system with four aisles and a conveyor belt has similar productivity to an RMFS with 50 robots. They also find that carefully determining the number of picking stations is essential to maintain a high picker utilisation while avoiding congestion. Guan and Li [66] derive association rules from historical demand data to decide which items to store together on a shelf, to maximise item similarity (the probability that items from the same shelf are ordered together) within the context of scattered storage. They propose a non-linear MIP and a genetic algorithm to solve it, and they obtain a higher item similarity of about 35%, but their results are not converted to actual productivity gains through simulations. Weidinger et al. [53] define a rack assignment problem for a batch of orders (static scheduling). Their objective is to assign each returning shelf to a storage location, with the surrogate objective of minimising the total loaded distance travelled by the robots. Because they consider the arrival time of each rack at the picking station as a known variable, they do not need to account for robots individually. They devise a variant of an adaptive

large neighbourhood search to solve their model and evaluate their approach in terms of total travelling time, for which they obtain better results compared to their baseline. Like [27], they note the good performance of the shortest leg decision rule, which operates completely online and presents an average cycle time that is only 3.5% higher than their method. In [58], the authors introduce RAWSim-O, a very detailed discrete event simulator that realistically reproduces operations taking place in an RMFS. Merschformann et al. [13] use this simulation framework to evaluate many combinations of decision rules. They test decision rules for decision problems such as picking and replenishment station assignment, pod selection, or storage assignment. Their study yields significant performance differences between combinations, which emphasises the importance of careful selection for each application. They also note the importance of a proper picking station assignment rule. The cross-dependencies existing between some rules suggests potential benefit in investigating integrated approaches.

Considering the variety of operational subproblems, the uncertainty about demand and operations completion times, as well as the dynamic nature of the operations that should respond in real time to new orders, we propose a mathematical framework that formalises the operational decision-making in such a dynamic stochastic setting. We follow the stochastic optimisation framework of Powell [63] with the objective of guiding future research on integrated approaches.

4.4 Formulation

Before presenting the complete dynamic model in Section 4.4.2, we first describe the key elements considered in the framework and the types of operational decisions we consider.

4.4.1 Modelling framework

As explained in the presentation of RMFSs in Section 4.2, the operations of a robot consist of a succession of dual command cycles. These cycles require several decisions that we describe here. Since the system is very dynamic and complex, any decision policy needs to be tested through simulation. For this reason, our modelling has strong similarities with a discrete event simulator. First, a decision needs to be made, not at regular time steps, but when a robot becomes available for a new task. We refer to this as an *event*.

Because we do not consider batching of orders, new orders are revealed online and need to be included in the pool of orders to fulfil right away. Note that we consider one-line orders only (one type of item, possibly several units of it) because most e-commerce orders present very few lines. Dealing with multi-line orders would entail additional considerations that are not

accounted for in our model. If one wanted to adapt our model to treat multi-line orders, we would suggest starting by splitting the orders into single-line orders. Additional constraints and variables would be required, particularly to treat two key elements : if consolidation is done at the picking station, the split orders need to be processed at the same station ; regardless of where the consolidation occurs, the different parts of a split order need to be retrieved in a short time span, to avoid saturating buffers and to validate the fulfilment of the order.

Importantly, some papers like [13] consider a constant order backlog. Here, we do not assume a constant backlog size, but we assume that an order is always waiting to be fulfilled. This situation makes the throughput of the system entirely depend on the planning decisions. This assumption can be justified by a usual high volume of orders in e-commerce, or for instance, the possibility to send a robot for maintenance or recharge while idle. However, if this assumption was to be relaxed, an event may be triggered by a new order entering the system, instead of an available robot. In this case, it may become interesting to batch multiple robots to allocate one or several orders. This consideration will remain out of the scope of the modelling in this work.

Furthermore, we do not consider the path planning problem, which consists of defining the exact movement of robots within the warehouse. However, because the travelling time taken by a robot may vary due to congestion, we consider stochastic travelling times in the general case. As such, we cannot know in advance exactly when a task will be completed. Like all the papers on real-time planning for RMFS reviewed in Section 3, we assume all tasks to be non-preemptive, which means that when a robot is assigned to a task it cannot be interrupted before completion, even if a better opportunity occurs.

At a picking station, a human operator can only pick items from one shelf at a time. When a robot arrives at a picking station, it may have to wait in a queue, following a first-in, first-out procedure. Note that in the model, we do not explicitly represent the replenishment of shelves, but we do consider a notion of stock levels. Instead, we suggest that a replenishment task could be accounted for in our modelling by considering orders of negative quantities. Since stations are typically fully dedicated to either replenishment or retrieval tasks, simple constraints about which station can process a specific order can be added, but those are not explicitly mentioned in the model.

While dual command cycles (consecutive storage and retrieval tasks) are the standard sequences of tasks for a robot, we also define less frequent *opportunistic tasks*. The purpose of an opportunistic task is to avoid a trip to the storage area if the shelf carried by a robot could be used again to fulfil another order. We distinguish two types of opportunistic tasks.

First, when a robot is leaving a picking station, it can go straight to the end of the waiting line of the same or another picking station. In this case, the robot saves the travelling time to the storage area and back. The second type of opportunistic task has the robot in place to deal immediately with another order. In this case, the robot saves both the travelling time and waiting time at a picking station.

Figure 4.1 presents the decision tree that guides decision-making when an event occurs. If the robot is located at a *picking station*, two types of decisions can be made : a storage task or an opportunistic task. On the other hand, if the robot is in the storage area, the robot must perform a retrieval task, which consists of choosing which order to fulfil, from which shelf, and to which picking station. To elaborate those decisions and their consequences, we first define useful sets and parameters. We will then describe the state variables along with exogenous information (random variables), the decisions that need to be made, the transition function, and the cost function.

4.4.2 Sets

The following sets, as well as the parameters described in Section 4.4.3, are fixed and determined by the warehouse design. They are not variables that will evolve over time.

$\mathcal{I}$ = set of items that can be found in the warehouse.
$\mathcal{L}$ = set of storage locations.
$\mathcal{S}$ = set of storage shelves.
$\mathcal{S}_i$ = subset of shelves $\subset \mathcal{S}$ carrying item $i \in \mathcal{I}$.
$\mathcal{R}$ = set of automated robots.
$\mathcal{P}$ = set of picking stations.

All item types stored in the warehouse are denoted by $\mathcal{I}$. Storage locations $\mathcal{L}$ define locations where a shelf can be stored in the warehouse. Those shelves are defined by set $\mathcal{S}$, and $\mathcal{S}_i$ identifies the subset of shelves containing a specific item $i \in \mathcal{I}$. Robots $\mathcal{R}$ represent the small automated robots that can store and retrieve shelves. Human operators are located at picking stations represented by set $\mathcal{P}$.

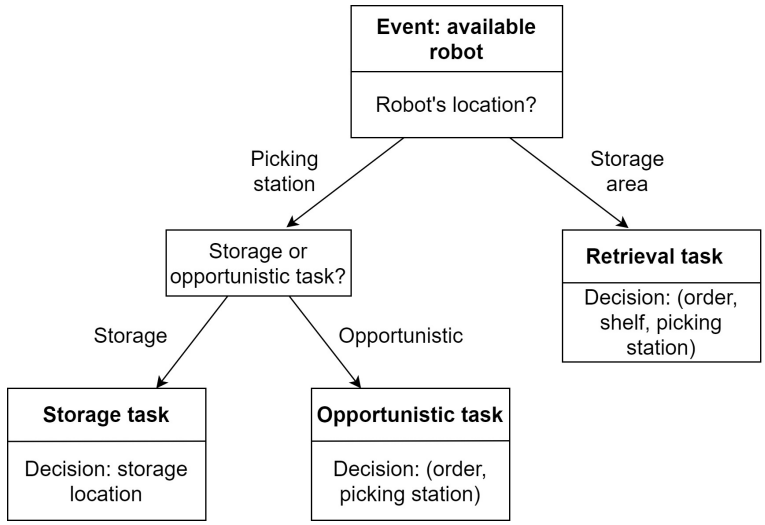

Figure 4.1 Decision tree

4.4.3 Parameters

d_o = deadline of order $o \in \mathcal{O}^k$ (state variable, see Section 4.4.4).
$cost_o$ = penalty cost for tardiness of order $o \in \mathcal{O}^k$ (expressed per late time unit).
$\overline{time}_{i,j}$ = expected time for a robot to travel from point i to point j ; $(i, j) \in (\mathcal{L} \cup \mathcal{P})^2$.
$\overline{time}_o$ = expected time required to pick all the items from order $o \in \mathcal{O}^k$ by an operator.
i_o = item type $\in \mathcal{I}$ of order $o \in \mathcal{O}^k$.
q_o = quantity of units of item type $i_o \in \mathcal{I}$ of order $o \in \mathcal{O}^k$.

Because of different levels of priority, such as with *Prime vs Standard* service for Amazon, each order can be associated with distinct penalty costs for tardiness $cost_o$ with respect to deadline d_o. $\overline{time}_{i,j}$ represents the estimated, or expected time needed by a robot to travel from location i to location j ; these locations can either be storage locations or picking stations. Depending on the application, if travelling times are deterministic, this value is simply an estimated time. Otherwise, with stochastic travelling times, the actual time taken by the robot is uncertain, but the expected value $\overline{time}_{i,j}$ may still be useful for decision-making. A human operator needs, on average, $\overline{time}_o$ to pick the items of order o from a shelf ; it may depend on the number of items to pick, the weight of the item, etc. i_o denotes the item type of order o. While we assume that orders are single-line orders, we still define q_o as the quantity of units of item type i_o to be picked. This allows for an order to contain more than one unit of the same item but also, should an order be a replenishment task, we could have $q_o < 0$.

4.4.4 State variables

The state of the warehouse at a given time step describes all of the relevant information needed to make operational decisions : the location of shelves and robots, availability of resources, orders to fulfil, etc. Following the framework of Powell [63], the state of the system at a given time step k is denoted by S_k and is decomposed into three entities : the *physical state* variables R_k, the *other information* variables I_k, and the *belief state* variables B_k. We detail the role and the components of these entities below.

$$S_k = (R_k,\, I_k,\, B_k)$$

Physical state

The physical state defines all physical characteristics of the warehouse, such as the location and availability of the shelves, the robots, and others. We decompose the physical state as $R_k = (t^k,\, \mathcal{L}^k_{\mathcal{S}},\, \mathcal{T}^k_{\mathcal{S}},\, stock^k_{\mathcal{S},\mathcal{I}},\, \mathcal{T}^k_{\mathcal{L}},\, z^k,\, \mathcal{L}^k_{\mathcal{R}},\, \mathcal{S}^k_{\mathcal{R}},\, \mathcal{T}^k_{\mathcal{R}},\, \mathcal{O}^k_{\mathcal{R}},\, \mathcal{R}^k_{\mathcal{P}})$, of which the components are described below.

Running time

$$t^k = \text{current time corresponding to time step } k.$$

In our model, a time step k corresponds to an event when a decision must be made, as in a discrete event simulation. The time interval between two such events can be highly variable. We define the state variable t^k as the time of event k.

Shelves

$$\begin{aligned}
l^k_s &= \text{location} \in \mathcal{L} \text{ of shelf } s \in \mathcal{S}\,;\ \mathcal{L}^k_{\mathcal{S}} = \{l^k_s \mid s \in \mathcal{S}\}.\\
\tau^k_s &= \text{time at which shelf } s \in \mathcal{S} \text{ becomes available}\,;\ \mathcal{T}^k_{\mathcal{S}} = \{\tau^k_s \mid s \in \mathcal{S}\}.\\
stock^k_{s,i} &= \text{stock level of item type } i \in \mathcal{I} \text{ on shelf } s \in \mathcal{S}\,;\\
&\quad\ stock^k_{\mathcal{S},\mathcal{I}} = \{stock^k_{s,i} \mid s \in \mathcal{S}, i \in \mathcal{I}\}.
\end{aligned}$$

A shelf $s \in \mathcal{S}$ is currently assigned to location $l^k_s \in \mathcal{L} \cup \mathcal{P}$ and became or will become available by time τ^k_s, as known by time t_k. In this context, 'available' means the shelf (or location, robot, etc.) can be used right away. At time t^k, even if $l^k_s \in \mathcal{L}$, shelf s may not be available yet, but will be at time $\tau^k_s > t^k$, because it may not have reached its storage location. By

convention, we set $\tau_s^k = +\infty$ when the arrival time has not been revealed yet or if the shelf is currently not assigned to a location. Note that in the case of stochastic travelling times, τ_s^k is only known when the shelf has been replaced, so its value will remain set to $+\infty$ until then. However, in a deterministic setting, the arrival time can be anticipated as soon as the storage decision is made. The stochastic case is presented here, but the details around the *belief state* variables will be similar to a deterministic setting. The variable $stock_{s,i}^k$ denotes the level of stock of item type i currently available on shelf s at time t_k (the current stock level may differ if the shelf was allocated to an order but not yet processed at the picking station). Depending on the number of items of type i required by an order, a shelf which contains this item may not be usable. Indeed, we consider that all items from an order must be retrieved from a single shelf. To relax this assumption and similar to multi-line orders, one may look into splitting the order into single-item orders (see Section 4.4.1).

Locations

τ_l^k = time at which location $l \in \mathcal{L}$ is available for storage ; $\mathcal{T}_{\mathcal{L}}^k = \{\tau_l^k \mid l \in \mathcal{L}\}$.

A location l is available for storage by time τ_l^k. This means that if $\tau_l^k \leq t^k$, the location is currently free. If $\tau_l^k = +\infty$, the location is not available for any planned horizon or a robot is on its way but has not yet retrieved the shelf.

Robots

z^k = robot $\in \mathcal{R}$ available for a task.
l_r^k = location $\in \mathcal{L} \cup \mathcal{P}$ of robot $r \in \mathcal{R}$; $\mathcal{L}_{\mathcal{R}}^k = \{l_r^k \mid r \in \mathcal{R}\}$.
s_r^k = shelf $\in \mathcal{S}$ carried by robot $r \in \mathcal{R}$; $\mathcal{S}_{\mathcal{R}}^k = \{s_r^k \mid r \in \mathcal{R}\}$.
τ_r^k = time at which robot $r \in \mathcal{R}$ is available ; $\mathcal{T}_{\mathcal{R}}^k = \{\tau_r^k \mid r \in \mathcal{R}\}$.
o_r^k = order $\in \mathcal{O}$ that is currently processed by robot $r \in \mathcal{R}$; $\mathcal{O}_{\mathcal{R}}^k = \{o_r^k \mid r \in \mathcal{R}\}$.

Robot $z^k \in \mathcal{R}$ denotes the robot currently available for a task (defining an event). A robot $r \in \mathcal{R}$ is currently located at location l_r^k, is carrying shelf s_r^k and will be available next at time τ_r^k. Finally, o_r^k denotes which order robot r is currently processing.

Picking stations

$\mathcal{R}_p^k$ = set of robots $\subset \mathcal{R}$ assigned to picking station $p \in \mathcal{P}$; $\mathcal{R}_{\mathcal{P}}^k = \{\mathcal{R}_p^k \mid p \in \mathcal{P}\}$

Because a human operator needs $time_o$ to pick the items of an order o, the robots wait in a queue. Robots that are assigned to picking station p, and will thus wait in the same queue, are identified by $\mathcal{R}_p^k$.

Other information variables

An essential aspect of dynamic operations in e-commerce comes from new orders continuously entering the system. At a given time step, a set of orders to fulfil is known, but new orders can arrive at any moment. We represent this set of revealed orders as a state variable $\mathcal{O}^k$ of type *other information* as it evolves exogenously even if still influenced by the previous selection of orders to fulfil. Since it is the only variable of this type, we have : $I_k = (\mathcal{O}^k)$.

Orders

$\mathcal{O}^k$ = orders yet to be fulfilled by time step k.

Belief state variables

Depending on the application, some extra information about distributions of random variables may be known and useful to make decisions. As such, they can be stored in the *belief state* B_k. As mentioned above, since we consider stochastic travelling times, we do not know in advance when a robot will reach its destination. However, we can use expected values while making decisions. We define such estimations, as well as information about demand distributions with $B_k = (\overline{\mathcal{T}}^k_{\mathcal{R}\cup\mathcal{S}\cup\mathcal{L}}, \overline{\mathcal{T}}^k_{\mathcal{R},\mathcal{P}}, \bar{\lambda}^k_{\mathcal{I}})$, described below.

$\bar{\tau}_j^k$ = believed (e.g., expected) time of availability of component $j \in \mathcal{R}\cup\mathcal{S}\cup\mathcal{L}$;
$\overline{\mathcal{T}}^k_{\mathcal{R}\cup\mathcal{S}\cup\mathcal{L}} = \{\bar{\tau}_j^k \mid j \in \mathcal{R}\cup\mathcal{S}\cup\mathcal{L}\}$.

$\bar{\tau}_{r,p}^k$ = believed time of arrival of robot r at picking station p ;
$\overline{\mathcal{T}}^k_{\mathcal{R},\mathcal{P}} = \{\bar{\tau}_{r,p}^k \mid r \in \mathcal{R},\, p \in \mathcal{P}\}$.

$\bar{\lambda}_i^k$ = average demand rate of item i, can be a Poisson average or any other distribution information ; $\bar{\lambda}^k_{\mathcal{I}} = \{\bar{\lambda}_i^k \mid i \in \mathcal{I}\}$.

$\bar{\tau}_j^k$ represents the believed, or estimated, time of availability of component j, which can be a robot, a shelf, or a location. This estimation can be useful when the true time of availability τ_j^k has not been revealed yet. It can be computed based on past data or on an expected travelling times model. For example, if component j represents a shelf, and if $t^k < \bar{\tau}_j^k < \tau_j^k = +\infty$, it means that shelf j is currently being brought back to a storage location, will be available

at an estimated time $\bar{\tau}_j^k$, but its exact arrival time is yet unknown. Similarly, $\bar{\tau}_{r,p}^k$ designates the estimated time of arrival of robot r at picking station p. This information can be useful to estimate the order of the robots in the waiting queue at the picking station, to determine the processing order. Finally, orders are arriving online, but it is reasonable to consider that some statistics about future orders are available based on current trends, forecasts, planned promotions, etc. As such, we consider here that the arrival of new orders can be reasonably well-modelled with Poisson distributions and that we have an estimate of their current averages $\bar{\lambda}_i^k$ that may evolve over time.

4.4.5 Exogenous information

Between stochastic travelling times and new orders entering the system, new information is continuously revealed, comprising most of the challenge of dynamic decision-making. We represent the exogenous information as a list of random variables W_{k+1} that are revealed during the operations : $W_{k+1} = ((\hat{r}^{k+1}, \hat{\tau}^{k+1}), \widehat{\mathcal{T}}_{\mathcal{L}}^{k+1}, \widehat{\mathcal{T}}_{\mathcal{R},\mathcal{P}}^{k+1}, \widehat{\Theta}^{k+1})$.

$(\hat{r}^{k+1}, \hat{\tau}^{k+1})$ = robot becoming available at the next time step and its time of availability. Those information are linked as $\hat{r}^{k+1} = \underset{r \in \mathcal{R}}{argmin}(\hat{\tau}_r^{k+1})$, but only the time of availability of robot $\hat{r}^{k+1}$ is revealed by time step $k+1$.

$\hat{\tau}_l^{k+1}$ = arrival time of a robot at retrieval location l. Between two decision time steps, a robot may have arrived at a retrieval location to retrieve a shelf, the storage location then becomes available ; $\widehat{\mathcal{T}}_{\mathcal{L}}^k = \{\hat{\tau}_l^k \mid l \in \mathcal{L}\}$.

$\hat{\tau}_{r,p}^{k+1}$ = arrival time of a robot r to picking station p. Between two decision time steps, a robot may have arrived at a picking station, in this case, its arrival time is revealed. Note that such a robot can very well be the next available robot $\hat{r}^{k+1}$; $\widehat{\mathcal{T}}_{\mathcal{R},\mathcal{P}}^k = \{\hat{\tau}_{r,p}^k \mid r \in \mathcal{R}, p \in \mathcal{P}\}$.

$\widehat{\Theta}^{k+1}$ = new orders that entered the system between time step k and $k+1$. Realisation of a random variable $\mathbf{\Theta_{t^k \to \hat{\tau}^{k+1}}}$ of newly revealed orders between time t^k and $\hat{\tau}^{k+1}$.

In our framework, an event k is defined by the need for a decision, corresponding to a robot requesting a new task. Because of stochastic travelling and processing times, this event is uncertain and is represented by the pair of random variables $(\hat{r}^{k+1}, \hat{\tau}^{k+1})$, which designates the robot asking for a decision and the time at which it happens. The availability of other robots is governed by distinct random variables $\hat{\tau}_{\mathbf{r}}^{\mathbf{k+1}}$, $r \in \mathcal{R}$, but only the realisation of the earliest available robot is revealed. Between two events, other relevant *episodes* (events that do not require a decision) may happen. One such episode is a robot reaching a retrieval location

l at time $\hat{\tau}_l^{k+1}$. This information is important because it frees the location for another storage task. Another similar episode is the arrival of a robot at a picking station $\hat{\tau}_{r,p}^{k+1}$ because it determines the order in the waiting queue. By convention, if such time has not been revealed during the two decision steps, its value $\hat{\tau}_{-}^{k+1}$ is set to $+\infty$. Finally, between the times of two events, new orders may be revealed in the system. This set of new orders, which arrive between times t^k and t^{k+1}, is represented by a random variable $\mathbf{\Theta_{t^k \to t^{k+1}}}$ for which a realisation is denoted by $\widehat{\Theta}^{k+1}$.

4.4.6 Decisions

For a given state of the system S_k, a decision, or action, x_k must be made. Two options are possible, depending on the location of the available robot z^k.

If robot z^k is currently located at a picking station (state : $l_{z^k}^k \in \mathcal{P}$, see Eq. 4.1), it is carrying a shelf, so its first option is to store the shelf back into the warehouse (storage task, $x_k \in X_k^1$). A decision about which location l to store the shelf at must be made (Figure 4.2, (1)). Of course, this location needs to be available at the decision time : $\tau_l^k \leq t^k$. If travelling times were deterministic, one could anticipate the availability of a storage location and verify that l will be available by the time the robot reaches it : $\tau_l^t \leq t^k + \overline{time}_{l(z^k)^k,l}$.

The other option is called an *opportunistic task*. Instead of storing its shelf, a robot can reuse it right away to fulfil another order o at picking station p ($x_k \in X_k^2$). We distinguish two cases of such an opportunistic task. First, the robot can bring its shelf to the end of the waiting queue of either the same (Figure 4.2, (3)) or a different (Figure 4.2, (2)) picking station (case $p \in \mathcal{P}$). Second, the robot can remain at its current position (Figure 4.2, (4)) to deal immediately with another order o (case $p = l_{z^k}^{k*}$ which designates the current picking station of the robot with an * as an exponent).

$$
\begin{aligned}
&\textbf{if } l_{z^k}^k \in \mathcal{P} : \\
&\quad x_k \in X_k^1 \cup X_k^2 \\
&\quad X_k^1 = \{l \in \mathcal{L} \mid \tau_l^k \leq t^k\} \textbf{ (storage task)} \\
&\quad X_k^2 = \{(o,p) \mid o \in \mathcal{O}^k,\, p \in \mathcal{P} \cup \{l_{z^k}^{k*}\},\, s_{z^k} \in \mathcal{S}_{i_o},\, stock_{s_{z^k},i_o}^k \geq q_o\} \\
&\qquad \textbf{(opportunistic task)}
\end{aligned}
\tag{4.1}
$$

If robot z^k is available but located at a storage location (state : $l_{z^k}^k \in \mathcal{L}$, see Eq. 4.2), it is waiting for a retrieval task. The decision is threefold : which order $o \in \mathcal{O}^k$ to fulfil, from

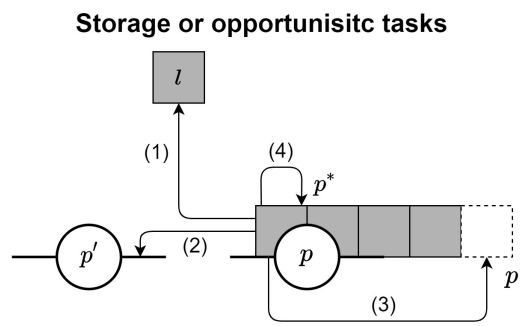

Figure 4.2 Possible decisions when a robot is located at a picking station

which shelf $s \in \mathcal{S}$, and by which picking station $p \in \mathcal{P}$. Of course, candidate shelves must contain the item of the order ($s \in \mathcal{S}_{i_o}$), have a sufficient stock level ($stock^k_{s,i_o} \geq q_o$), and must be available at decision time ($\tau^k_s \leq t^k$).

$$
\begin{aligned}
&\textbf{if } l^k_{z^k} \in \mathcal{L} \textbf{ (retrieval task)} : \\
&\quad x_k \in X^3_k \\
&\quad X^3_k = \{(o, s, p) \mid o \in \mathcal{O}^k,\ s \in \mathcal{S},\ p \in \mathcal{P},\ \tau^k_s \leq t^k,\ s \in \mathcal{S}_{i_o},\ stock^k_{s,i_o} \geq q_o\}
\end{aligned}
\tag{4.2}
$$

4.4.7 Transition function

The transition function S^M describes how the state variables are modified from S_k to S_{k+1} when a decision x_k is made and exogenous information W_{k+1} is revealed.

$$S_{k+1} = S^M(S_k,\ x_k,\ W_{k+1})$$

Note that to slightly lighten the notation, state variables that *do not* change between two consecutive time steps are not detailed. In this case, one must understand that these variables keep the same value at the next step.

First, for all states and decisions, the transition function starts with Algorithm 4.1. This part is straightforward : the robot available for a task is set to the newly revealed available robot (line 1, Alg. 1), the running time to the time at which this robot becomes available (line 2, Alg. 1), and the newly revealed orders between times t^k and t^{k+1} are added to the pool of orders to fulfil (line 3, Alg. 1). Then, if a location has been freed up during the time interval, its true and believed times of availability are updated (lines 4-6, Alg. 1). Similarly, if a robot has reached a picking station, its believed reaching time is updated (line 7-10, Alg. 1).

Algorithm 4.1 Transition function - general update

$z^{k+1} = \hat{r}^{k+1}$
$t^{k+1} = \hat{\tau}^{k+1}$
$\mathcal{O}^{k+1} = \mathcal{O}^k \cup \widehat{\Theta}^{k+1}$
for $l \in \mathcal{L}$ **do**
    if $\hat{\tau}_l^{k+1} < +\infty$ **then**
        $\tau_l^{k+1} = \bar{\tau}_l^{k+1} = \hat{\tau}_l^{k+1}$
for $r \in \mathcal{R}$ **do**
    for $p \in \mathcal{P}$ **do**
        if $\hat{\tau}_{r,p}^{k+1} < +\infty$ **then**
            $\bar{\tau}_{r,p}^{k} = \hat{\tau}_{r,p}^{k+1}$

The rest of the transition function depends, again, on the location of the available robot.

Transition - robot located at picking station ($l_{z^k}^k \in \mathcal{P}$, Algorithm 4.2)

When a robot is located at a picking station, it can either perform a storage task or an opportunistic task.

If a storage task is chosen (line 1, Alg. 2), then the decision concerns a storage location $x_k = l$. First, the locations of both the robot z_k and its shelf $s_{z_k}^k$ are set to location l (line 2, Alg. 2). Location l is now reserved with no information about when it will become available again, so its true and believed times of availability are both set to ∞ (line 3, Alg. 2). Because of uncertain travelling times, the time of availability of robot $\tau_{z^k}^{k+1}$ is unknown and set to ∞ (line 3, Alg. 2). However, its believed time of availability (and the one of its shelf) can be estimated as the current time plus the expected travelling time from the picking station to the storage location (line 4, Alg. 2). The robot is leaving picking station $l_{z^k}^k$ so its estimated time of arrival at this picking station is set to ∞ (line 5, Alg. 2), and the robot is removed from the robots assigned to the station (line 6, Alg. 2).

If the robot performs an opportunistic task instead, a new order o and a picking station p are selected (line 7, Alg. 2). The locations of the robot and its shelf are set to the picking station p (line 8, Alg. 2). The new order will pick items from the shelf, so the stock level of the shelf in items i_o is updated (line 9, Alg. 2). Order o is removed from the list of remaining orders to fulfil (line 10, Alg. 2) and the robot is assigned to this order (line 11, Alg. 2). By the next time step, if the same robot z^k has not yet arrived at station p and the decision was to send the robot at the end of a waiting queue ($p \neq l_{z^k}^{k*}$), its estimated time of arrival to station p is set to the running time plus the average travelling time between the two stations

(line 14, Alg. 2). Then, if the new picking station is different from the current one, robot z^k is removed from the robots assigned to station $l^k_{z^k}$ (line 16, Alg. 2) and added to the ones of station p (line 18, Alg. 2) ; its estimated time of arrival to station $l^k_{z^k}$ is also set to $+\infty$ (line 17, Alg. 2).

Finally, in both cases, if more robots are still assigned to station $l^k_{z^k}$ (line 19, Alg. 2), one can identify the next robot to be processed as the robot r that arrived the earliest (line 19, Alg. 2). If this robot r is not the one available at next time step (which would already be revealed), its believed time of availability is set to the running time plus the average time needed by the human operator to pick the items of its assigned order $\overline{time}_{o^k(r)}$ (line 22, Alg. 2).

Algorithm 4.2 Robot located at a picking station ($l^k_{z^k} \in \mathcal{P}$)

1: **if** $x_k = l \in X^1_k \subset \mathcal{L}$ **then** ▷ Storage task
2: $\quad l^{k+1}_{z^k} = l^{k+1}_{s^k(z^k)} = l$
3: $\quad \tau^{k+1}_l = \bar{\tau}^{k+1}_l = \tau^{k+1}_{z^k} = +\infty$
4: $\quad \bar{\tau}^{k+1}_{z^k} = \bar{\tau}^{k+1}_{s^k(z^k)} = t^k + \overline{time}_{l(z^k)^k,l}$
5: $\quad \bar{\tau}^{k+1}_{z^k,l^k(z^k)} = +\infty$
6: $\quad \mathcal{R}^{k+1}_{l^k(z^k)} = \mathcal{R}^k_{l^k(z^k)} \setminus z^k$
7: **else if** $x_k = (o,p) \in X^2_k \subset (\mathcal{O}^k, \mathcal{P} \cup \{l^{k*}_{z^k}\})$ **then** ▷ Opportunistic task
8: $\quad l^{k+1}_{z^k} = l^{k+1}_{s^k(z^k)} = p$
9: $\quad stock^{k+1}_{s^k(z^k),i_o} = stock^k_{s^k(z^k),i_o} - q_o$
10: $\quad \mathcal{O}^{k+1} = \mathcal{O}^k \setminus o$
11: $\quad o^{k+1}_{z^k} = o$
12: $\quad$ **if** $\hat{\tau}^{k+1}_{z^k,p} = +\infty$ **then**
13: $\quad\quad$ **if** $p \in \mathcal{P}$ **then**
14: $\quad\quad\quad \bar{\tau}^{k+1}_{z^k,p} = t^k + \overline{time}_{l^k(z^k),p}$
15: $\quad$ **if** $p \in \mathcal{P} \setminus l^k_{z^k}$ **then**
16: $\quad\quad \mathcal{R}^{k+1}_{l^k(z^k)} = \mathcal{R}^k_{l^k(z^k)} \setminus z^k$
17: $\quad\quad \bar{\tau}^{k+1}_{z^k,l^k(z^k)} = +\infty$
18: $\quad\quad \mathcal{R}^{k+1}_p = \mathcal{R}^k_p \cup z^k$
19: **if** $\mathcal{R}^{k+1}_{l^k(z^k)} \neq \varnothing$ **then** ▷ Common to both tasks
20: $\quad$ **let** $r = \underset{r' \in \mathcal{R}}{argmin}(\bar{\tau}^{k+1}_{r',l^k(z^k)})$
21: $\quad$ **if** $r \neq \hat{r}^{k+1}$ **then**
22: $\quad\quad \bar{\tau}^{k+1}_r = t^k + \overline{time}_{o^k(r)}$

Transition - robot located in storage area ($l^k_{z^k} \in \mathcal{L}$, Algorithm 4.3)

When a robot is available at a storage location, it needs to perform a retrieval task which consists in making a decision $x_k = (o, s, p)$ of an order o to fulfil, from shelf s, at picking station p.

First, both the true and believed times of availability of the shelf just dropped by the robot are now set to the running time (line 1, Alg. 3). The robot is now carrying the selected shelf s (line 2, Alg. 3). The true and believed times of availability of this shelf, as well as the true time of availability of the robot, are all set to $+\infty$ (line 3, Alg. 3) because the estimated time of when the items will be picked is hard to obtain at this stage (even if a rough estimate could be considered). The locations of the robot and shelf s become station p (line 4, Alg. 3) and the robot is added to the set of robots assigned to the picking station (line 5, Alg. 3). The stock level of the shelf in item type i_o is updated (line 6, Alg. 3). The order o is removed from the set of orders to fulfil (line 7, Alg. 3) and the order is assigned to robot z^k (line 8, Alg. 3). If the robot has not reached the retrieval location by time step $k+1$, the believed time of availability of storage location l^k_s is estimated to be the running time plus the travelling time between the drop-off and retrieval locations (lines 9-10, Alg. 3). Finally, if the robot has not arrived at the picking station by time step $k+1$, the believed time of arrival of robot z^k at station p is calculated as the running time plus the travelling time from drop-off to retrieval locations and retrieval location to picking station (line 11-12, Alg. 3).

Algorithm 4.3 Retrieval task ($l^k_{z^k} \in \mathcal{L}, x_k = (o, s, p) \in X^3_k \subset (\mathcal{O}^k, \mathcal{S}, \mathcal{P})$)

1: $\tau^{k+1}_{s^k(z^k)} = \bar{\tau}^{k+1}_{s^k(z^k)} = t^k$
2: $s^{k+1}_{z^k} = s$
3: $\tau^{k+1}_s = \bar{\tau}^{k+1}_s = \bar{\tau}^{k+1}_{z^k} = +\infty$
4: $l^{k+1}_{z^k} = l^{k+1}_s = p$
5: $\mathcal{R}^{k+1}_p = \mathcal{R}^k_p \cup z^k$
6: $stock^{k+1}_{s,i_o} = stock^k_{s,i_o} - q_o$
7: $\mathcal{O}^{k+1} = \mathcal{O}^k \setminus o$
8: $o^{k+1}_{z^k} = o$
9: **if** $\hat{\tau}^{k+1}_{l^k_s} = +\infty$ **then**
10: $\quad \bar{\tau}^{k+1}_{l^k_s} = t^k + \overline{time}_{l^k(z^k), l^k_s}$
11: **if** $\hat{\tau}^{k+1}_{z^k, p} = +\infty$ **then**
12: $\quad \bar{\tau}^{k+1}_{z^k, p} = t^k + \overline{time}_{l^k(z^k), l^k_s} + \overline{time}_{l^k_s, p}$

4.4.8 Cost function

The cost function defines the cost contribution of a decision to the global objective that needs to be optimised. While the previous parts of the model are quite universal, the objective function depends on the Key Performance Indicator (KPI) the decision-maker favours most. Merschformann et al. [13] present several relevant KPIs, such as the order throughput rate, the order turnover time, the travelling time, the fraction of late orders and others. To illustrate, we present here two objective contributions related to travelling times and deadlines. The first one considers the full cycle time in a fully stochastic setting (as the rest of the model). The second one only considers the travelling time of the robots in a setting where travelling times are deterministic. The reason for this double illustration is that to truly assess the complete cycle time of a robot in the fully stochastic case, we need to wait for the robot to have finished its cycle, which makes the contribution of each decision less perceptible. In both cases, Figure 4.3 illustrates the time components that compose a complete cycle.

Complete cycle time – stochastic travelling times

In this case, the time taken by a robot to perform a complete cycle is considered in the objective, which includes the travelling time as well as any waiting time spent in a queue at a picking station. Because the real time taken by a robot to complete a task is only revealed once the task has been completed, its contribution to the objective function is likewise delayed. For this reason, we add a physical state variable for each robot, representing the time at which it started its task :

$$\varphi_r^k = \text{time at which robot } r \in \mathcal{R} \text{ started its new task}\,;\ \varphi_{\mathcal{R}}^k = \{\varphi_r^k \mid r \in \mathcal{R}\}$$

And we update this state variable every time a robot is assigned to a new task. We add at the end of Algorithm 4.1, line 11 :

Algorithm 4.4 Extra line in Algorithm 1

11: $\varphi_{z^k}^{k+1} = t^k$

Now, in every state, the time taken to perform the *previous* task can be computed by taking the difference between the current time t^k and the time the robot started this task $\varphi_{z^k}^k$ (Eq. 4.3). Also, if the robot is located at a picking station, an order was just completed, and a penalty cost is applied to penalize, if necessary, a violated deadline : $cost_{o^k(z^k)} \times max[0, t^k - d_{o^k(z^k)}]$.

For a given state, decision and exogenous information, we denote the objective contribution by : $C(S_k, x_k, W_{k+1})$

$$
\begin{aligned}
&\textbf{if } l^k_{z^k} \in \mathcal{P} \textbf{ (Robot located at a picking station)} : \\
&\qquad C(S_k, x_k, W_{k+1}) = t^k - \varphi^k_{z^k} + cost_{o^k(z^k)} \times max[0, t^k - d_{o^k(z^k)}] \\
&\textbf{if } l^k_{z^k} \in \mathcal{L} \textbf{ (Robot located at a storage location)} : \\
&\qquad C(S_k, x_k, W_{k+1}) = t^k - \varphi^k_{z^k}
\end{aligned}
\tag{4.3}
$$

Travelling time only – deterministic travelling times

We present the objective of the deterministic travelling time to have a more intuitive representation of the objective contribution. In Eq. 4.4, we distinguish the different types of decisions. In the case of a storage task or an opportunistic task, the costs are similar : the travelling time from the current picking station to either the storage location or another picking station (if the robot stays in place to deal immediately with another order, this cost is 0), and a penalty cost for violated deadlines. If the robot performs a retrieval task, the cost corresponds to the interleaving time between the drop-off and the retrieval locations plus the time between the retrieval location and the picking station.

$$
\begin{aligned}
&\textbf{if } l^k_{z^k} \in \mathcal{P} \text{ and } x_k = l \in X^1_k \subset \mathcal{L} \textbf{ (Storage task)} : \\
&\qquad C(S_k, x_k, W_{k+1}) = \overline{time}_{l(z^k)^k, l} + cost_{o^k(z^k)} \times max[0, t^k - d_{o^k(z^k)}] \\
&\textbf{if } l^k_{z^k} \in \mathcal{P} \text{ and } x_k = (o, p) \in X^2_k \subset (\mathcal{O}^k, \mathcal{P} \cup \{l^{k*}_{z^k}\}) \textbf{ (Opportunistic task)} : \\
&\qquad C(S_k, x_k, W_{k+1}) = \overline{time}_{l(z^k)^k, p} + cost_{o^k(z^k)} \times max[0, t^k - d_{o^k(z^k)}] \\
&\textbf{if } l^k_{z^k} \in \mathcal{L} \text{ and } x_k = (o, s, p) \in X^3_k \subset (\mathcal{O}^k, \mathcal{S}, \mathcal{P}) \textbf{ (Retrieval task)} : \\
&\qquad C(S_k, x_k, W_{k+1}) = \overline{time}_{l^k(z^k), l^k_s} + \overline{time}_{l^k_s, p}
\end{aligned}
\tag{4.4}
$$

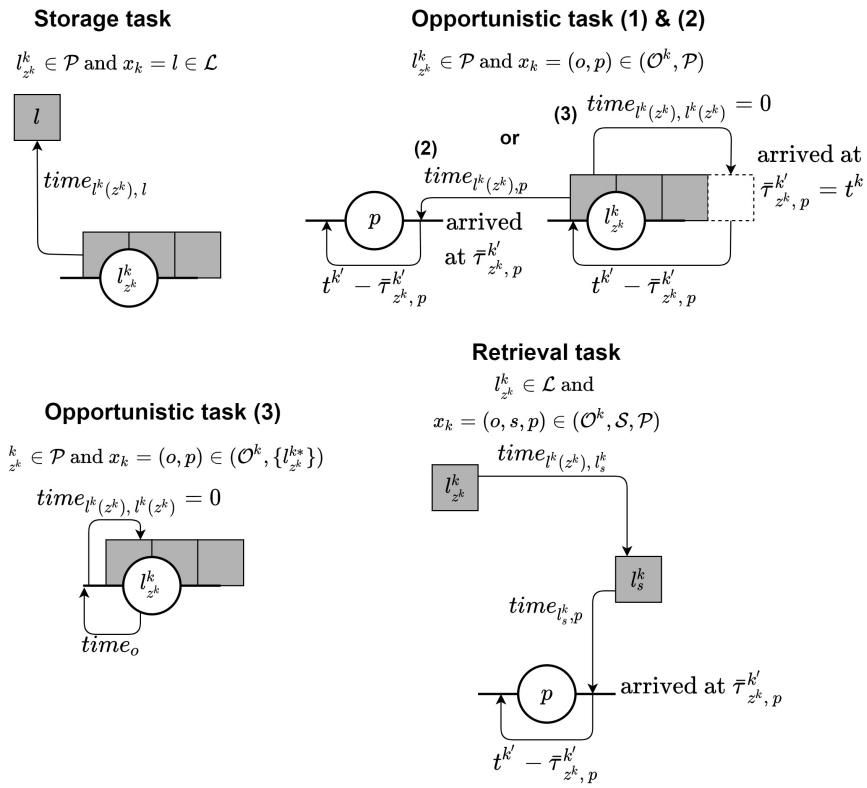

Figure 4.3 Illustration of the cost components in the complete cycle time

Global objective

Finally, the global objective is, for every state, to make the decision that minimises the global cost of operations. This is defined recursively by Eq. 4.5, equivalent to Bellman equations.

$$\min_{\pi} \mathbb{E}_{S_0} \mathbb{E}_{W_1 \ldots W_{+\infty} | S_0} \left\{ \sum_{k=0}^{+\infty} \alpha^k \, C_k(S_k,\, X_k^{\pi}(S_k),\, W_{k+1}) \mid S_0 \right\}, \tag{4.5}$$

where α is a discount rate $\in \,]0\,;1[$ and :

$$S_{k+1} = S^M(S_k,\, X_k^{\pi}(S_k),\, W_{k+1}) \tag{4.6}$$

More precisely, the objective is to define a policy π that minimises the expected infinite sum of discounted cost contributions $C_k(S_k,\, X_k^{\pi}(S_k),\, W_{k+1})$, considering the recursive relationship between two consecutive states defined by Eq. 4.6.

We may also prefer to not consider any discount rate in the objective function, for instance to properly minimise the expected average cycle time. In this case, we can define the global

objective as in Eq. 4.7, which averages the infinite sum of cost contributions.

$$\min_{\pi} \lim_{h \to +\infty} \mathbb{E}_{S_0} \mathbb{E}_{W_1 \dots W_h | S_0} \left\{ \frac{1}{h} \sum_{k=0}^{h} C_k(S_k,\, X_k^{\pi}(S_k),\, W_{k+1}) \mid S_0 \right\} \tag{4.7}$$

4.5 Simulation study

Because the subproblem of storage allocation is recurrent both in traditional AS/RSs and also in RMFSs, we propose illustrating a simple usage of our framework by comparing the performance of the four most common storage decision rules. The objective of this section is neither to propose a new method, nor a fully integrated simulation study, but to compare storage policies that should be considered as baselines when developing new methods. We study the performance of those baselines in terms of the two KPIs presented in Section 4.4.8 : the travelling time and the full cycle time. The baselines are tested in isolation under simplification assumptions that are presented below.

4.5.1 Assumptions

To study storage allocation baselines in isolation, we make several simplifications. First, we consider a single picking station. A type of item is stored on exactly one shelf, which only contains this item, and replenishment is ignored. Both travelling and picking times are deterministic. Revealed orders are processed following their deadlines, with the condition that the required item is on a shelf that is not currently being carried by another robot. Because this online scheduling is not optimised, delay penalties in the objective functions presented in the model are removed. Only opportunistic tasks of type 1 (where the robot goes straight to the end of the waiting queue at a picking station) are considered, and they are enforced as often as possible (if the next order needs the same shelf that the robot is currently carrying). Also, because the next order assigned to a robot is an external decision, we can use this information at the time when a storage decision is made. To do so, we add an *other information* variable to the ones defined in Section 4.4.4, representing this order. Let us denote this state variable by o^k_{next}. Note that the only event that requires a decision is when a robot is available at the picking station : $l^k_{z^k} \in \mathcal{P}$, we can then omit the case of a retrieval task when $l^k_{z^k} \in \mathcal{L}$ in Eq. 4.2.

4.5.2 Storage allocation baselines

In this section, we express the storage decision rules within our modelling framework. Because we enforce opportunistic tasks of type 1 whenever possible, an actual storage task only occurs when $s^k_{z^k} \neq \mathcal{S}_{i(o^k_{\text{next}})}$ (here $\mathcal{S}_i$ only designates one shelf instead of a set), which means that the robot is not carrying the shelf containing the required item. Otherwise, the opportunistic task is performed by default, which consists of processing the next order o^k_{next} at picking station $p = 1$ (because there is only one picking station) : $x_k = (o^k_{\text{next}}, p = 1)$. We only represent this second case in the random storage policy.

Random storage

In random storage, the storage decision x_k is randomly drawn from the uniform distribution over the set of available locations at time t^k.

$$\begin{aligned}
&\textbf{if } s^k_{z^k} \neq \mathcal{S}_{i(o^k_{\text{next}})} : \\
&\qquad x_k \sim U(X^1_k) \\
&\qquad \textbf{where } X^1_k = \{l;\, l \in \mathcal{L},\, \tau^k_l \leq t^k\} \\
&\textbf{else :} \\
&\qquad x_k = (o^k_{\text{next}}, p = 1) \in (\mathcal{O}^k, \mathcal{P}) \subset X^2_k
\end{aligned} \tag{4.8}$$

Closest open location

The closest open location policy consists of selecting the available storage that is closest to the picking station as defined by Eq. 4.9.

$$\begin{aligned}
&\textbf{if } s^k_{z^k} \neq \mathcal{S}_{i(o^k_{\text{next}})} : \\
&\qquad x_k = \underset{l}{argmin} \left(time_{p=1,l} \mid l \in \mathcal{L},\, \tau^k_l \leq t^k \right)
\end{aligned} \tag{4.9}$$

Class-based storage

In class-based storage, a shelf is randomly stored within a class based on its turnover rate. Classes are distributed by increasing distances from the picking station and shelves with the highest turnovers are assigned to the closest classes. Because here a shelf contains exactly one item type, which can only be found on this shelf, the turnover rate of a shelf s corresponds

to that of its item type ($\bar{\lambda}_i^k$; $s = \mathcal{S}_i$). Eq. 4.10 randomly draws the storage location from a uniform distribution over the set of available locations within the class of shelf $s_{z^k}^k$. The item contained on the shelf is $i(o^k(z^k))$, which gives the turnover rate of the shelf Class($s_{z^k}^k$).

$$\begin{aligned}
&\textbf{if } s_{z^k}^k \neq \mathcal{S}_{i(o_{\text{next}}^k)} : \\
&\quad x_k \sim U(\text{Class}(s_{z^k}^k)) \\
&\quad \textbf{where } \text{Class}(s_{z^k}^k) \text{ defines the open locations of the class associated with} \\
&\quad \text{shelf } s_{z^k}^k \text{ based on its turnover rate } \bar{\lambda}_{i(o^k(z^k))}^k
\end{aligned} \tag{4.10}$$

Shortest leg

For the shortest leg decision rule, Eq. 4.11 selects the open location that minimises the travelling time to the storage location plus the time from the storage location to the next retrieval location. The next shelf to retrieve for the robot at hand is $\mathcal{S}_{i(o_{\text{next}}^k)}$, and its location is $l' = l(\mathcal{S}_{i(o_{\text{next}}^k)})$. So, the selected location x_k is the one that minimises $time_{p=1,l} + time_{l,l'}$.

$$\begin{aligned}
&\textbf{if } s_{z^k}^k \neq \mathcal{S}_{i(o_{\text{next}}^k)} : \\
&\quad x_k = \underset{l}{argmin} \left(time_{p=1,l} + time_{l,l'} \mid l \in \mathcal{L},\, \tau_l^k \leq t^k,\, l' = l(\mathcal{S}_{i(o_{\text{next}}^k)}) \right)
\end{aligned} \tag{4.11}$$

4.5.3 Simulation study

In this section, we run simulations to evaluate the four decision rules presented above with the two cost functions of Section 4.4.8 : full-cycle time and travelling time with the average cost global objective (Eq. 4.7). We define a reasonably small warehouse storage area since we consider only one picking station, and we keep the setting parameters constant except the skewness parameter of the demand distribution that we describe below. Multiple picking stations could be considered without major changes, except for the class-based storage, which would require a careful zones definition, see, for instance, [46].

The storage area contains 108 storage locations for 103 shelves, and 8 robots are deployed. The speed of the robots is set to 0.6 m/s, accounting for acceleration, deceleration, loading, and unloading times. The picking time taken by the human operators is set to 5 seconds. The number of classes for the class-based storage policy is set to 6. Figure 4.4 gives a plan view of such a warehouse.

As in regular warehousing, some items are much more popular than others, which translates

into skewed demand distributions. We generate orders following the principle of an ABC curve proposed by Hausman et al. [20] : $G(x) = x^s$, for $0 < s \leq 1$, represents the ranked cumulative % demand versus % (x) of item types. The smaller the s, the more skewed the demand distribution. To every item type i a Poisson distribution of average $\bar{\lambda}_i = \left(\left(\frac{i}{m}\right)^s - \left(\frac{i-1}{m}\right)^s\right) \times \frac{n}{N}$ is assigned, where m is the number of item types, n the expected total number of orders over the time horizon and N the number of discretized periods. Orders are then generated online at each discretized period, for each item. Then, for every generated order k, a completion time δ_k (time before deadline) is randomly drawn from a uniform distribution of interval $[1; \alpha \times T]$, where α defines the tightness of the completion times. In this simulation study, we run the simulation for 24 hours with time steps of 60 seconds to generate new orders. Several values of skewness parameter s are tested and the tightness parameter α is set to 0.4. We also set $m = 103$, $n = 30,000$ and $N = 1440$ $(24\,h\,/\,60\,sec)$.

Table 4.1 presents the performance of the four decision rules in terms of travelling time for 8 values of the skewness parameter s. Note that, because of the stochasticity in the simulation process, each value presented in the table corresponds to the average over 100 runs. For each decision rule, one column presents the average travelling time, another one the corresponding standard deviation σ (over the 100 runs), and finally the relative gain compared to the random storage policy. First, we notice that the travelling time, for all storage policies, increases with the increase of parameter s. This is explained by the greater number of opportunistic tasks that can be performed with skewed distributions. Indeed, when an item is ordered much more often relative to others, the chances that two consecutive orders will require the same item are greater. Overall, the best-performing policy is the shortest leg (SL), with performance gains ranging from 10.97% to 11.40%. The only exception is for extremely skewed distributions ($s = 0.4$ and 0.5), for which class-based storage performs best with an improvement of 12.12% and 14.89%. However, the performance of class-based storage declines sharply when the value of s increases ; it even becomes worse than random when $s = 1$. Interestingly, SL's performance remains reasonably constant, with some extra performance gains for smaller s. This can be explained by the fact that often-required shelves are frequently moved and may end up occupying, by chance, locations closer to the picking station, which results in shorter cycle times. We note the poor global performance of the closest open location (COL) method that remains independent of the value of parameter s, with performance gains around 5%.

Figure 4.4 Simulation study - storage area

Table 4.1 Travelling times

Storage policy	Random		COL			Class-based			SL		
	t(s)	σ(s)	t(s)	σ(s)	g(%)	t(s)	σ(s)	g(%)	t(s)	σ(s)	g(%)
s=0.4	46.85	0.75	44.74	0.78	4.51	39.87	0.70	**14.89**	41.51	0.68	11.40
s=0.5	50.52	0.85	48.08	0.81	4.83	44.40	0.73	**12.12**	44.81	0.68	11.29
s=0.6	52.67	0.67	50.00	0.73	5.06	47.92	0.70	9.01	46.79	0.64	**11.15**
s=0.7	53.79	0.64	51.15	0.60	4.91	50.63	0.63	5.88	47.85	0.59	**11.04**
s=0.8	54.26	0.57	51.70	0.57	4.70	52.62	0.59	3.01	48.30	0.52	**10.97**
s=0.9	54.59	0.57	51.92	0.53	4.90	54.39	0.57	0.37	48.60	0.45	**10.97**
s=1.0	54.69	0.56	51.98	0.59	4.95	55.60	0.58	-1.67	48.68	0.47	**11.00**

Table 4.2 presents the same type of information but in terms of full cycle time, considering the waiting time spent by robots at the picking station as well. We note some relevant differences. First, as expected, all policies result in longer full-cycle times compared to travelling times only (which is a lower bound) ; robots take on average 10 seconds more to complete a full cycle. This simply means that robots are losing time waiting in the waiting queue. Then, while the cycle time for all policies still benefits from lower s values, the differences are less significant. This can come from opportunistic tasks that do not result in significant performance gains due to the waiting time at the picking station. The best-performing policies dependent upon s remain essentially the same as before. Interestingly, SL, the best overall policy, does not seem to benefit from smaller s values, compared to random. It may be that conveniently-located, high-turnover shelves do not result in faster cycles, once again because of robots waiting to be processed at the picking stations.

Table 4.2 Full cycle times

Storage policy	Random		COL			Class-based			SL		
	t(s)	σ(s)	t(s)	σ(s)	g(%)	t(s)	σ(s)	g(%)	t(s)	σ(s)	g(%)
s=0.4	55.60	0.64	53.51	0.66	3.76	50.06	0.56	**9.96**	50.77	0.51	8.70
s=0.5	58.65	0.78	56.11	0.71	4.32	53.43	0.61	8.90	53.20	0.57	**9.29**
s=0.6	60.50	0.61	57.63	0.66	4.75	56.23	0.60	7.06	54.69	0.54	**9.60**
s=0.7	61.46	0.58	58.64	0.55	4.58	58.54	0.56	4.76	55.55	0.52	**9.61**
s=0.8	61.86	0.54	59.12	0.52	4.44	60.29	0.53	2.54	55.94	0.47	**9.57**
s=0.9	62.18	0.52	59.29	0.49	4.65	61.90	0.52	0.44	56.20	0.40	**9.62**
s=1.0	62.26	0.52	59.33	0.54	4.71	62.97	0.54	-1.15	56.25	0.42	**9.65**

While this study remains simple, with an illustrative objective only, we can draw some relevant insights from the results. First, considering the performance of class-based and SL policies on the travelling time only, we understand the existence of a trade-off between strategically locating shelves based on the turnover rate of the items, to favour future accessibility, and minimising immediate cycle times. Then, with regards to the waiting time spent in the picking station's queue, we understand that time saved in travelling can be wasted.

4.6 Conclusions and future work

In this work, we presented a dynamic stochastic optimisation framework that models real-time operational decisions within a Robotic Mobile Fulfillment System. This new system of automated warehouses was developed specifically for the challenges of the e-commerce market. Among those challenges, online retailers need to fulfil orders extremely fast in a highly competitive environment. Thus, the necessity of considering orders as soon as they are revealed to the system. Coupled with the great flexibility of storage allocation of RMFSs, as well as the diverse operational decisions that need to be made, the global process results in a stochastic dynamic problem that is typically tackled by high-level decision rules. By formalising such a problem with a model, even though there is no obvious solving mechanism to be proposed, it will assist researchers in the development of more advanced methods to improve the performance of either specific subproblems in isolation or the integrated problem. Essential elements of the model are : stochastic demand (orders are revealed online) ; stochastic travelling and picking times for the robots and operators ; irregular decision-making time steps that follow times at which events occur, similar to a discrete event simulation ; definition of opportunistic tasks that model the possibility of combining two compatible orders to save travelling time ; a waiting queue for robots at picking stations. To illustrate the model, we propose a simple simulation study using storage allocation decision rules from the literature.

We transcribe these rules following the model's proposed notation and run experiments to evaluate their performance and gain insights about improvement opportunities. In particular, the Shortest Leg storage policy demonstrates its good performance in terms of travelling time. When the cycle time includes the waiting time at the picking station, distributing the arrival of the robots to avoid idle times appears to be essential to benefit from a good storage policy entirely.

In the future, based on the assessment of the storage decision rules illustration, we wish to build on the modelling framework to develop new storage policies other than traditional decision rules. We think that methods coming from approximate dynamic programming or reinforcement learning could be applicable here, to learn from repeated experiences and demand trends in order to improve performance. While considering storage allocation only, we understand that there exists a trade-off between minimising travelling times and avoiding saturation at picking stations. Finally, future research could also aim at proposing a systematic approach on how to model and consider multi-line orders, as well as path planning to avoid traffic congestion and collision, along with the real-time task allocation presented in this work.

CHAPITRE 5 ARTICLE 2 : E-COMMERCE WAREHOUSING : LEARNING A STORAGE POLICY

Adrien Rimélé[ab], Philippe Grangier[d], Michel Gamache[ac], Michel Gendreau[ab], Louis-Martin Rousseau[ab]

[a]Department of Mathematics and Industrial Engineering, Polytechnique Montréal, [b]CIRRELT, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, [c]GERAD, Group for Research in Decision Analysis, [d]IVADO Labs

Cet article a été soumis à *European Journal of Operational Research* le 11 janvier 2021.

ABSTRACT

E-commerce with major online retailers is changing the way people consume. The goal of increasing delivery speed while remaining cost-effective poses significant new challenges for supply chains as they race to satisfy the growing and fast-changing demand. In this paper, we consider a warehouse with a Robotic Mobile Fulfillment System (RMFS), in which a fleet of robots stores and retrieves shelves of items and brings them to human pickers. To adapt to changing demand, uncertainty, and differentiated service (e.g., prime vs. regular), one can dynamically modify the storage allocation of a shelf. The objective is to define a dynamic storage policy to minimise the average cycle time used by the robots to fulfil requests. We propose formulating this system as a Partially Observable Markov Decision Process, and using a Deep Q-learning agent from Reinforcement Learning, to learn an efficient real-time storage policy that leverages repeated experiences and insightful forecasts using simulations. Additionally, we develop a rollout strategy to enhance our method by leveraging more information available at a given time step. Using simulations to compare our method to traditional storage rules used in the industry showed preliminary results up to 14% better in terms of travelling times.

KEYWORDS

Decision processes ; Supply chain management ; E-commerce ; Storage Policy ; Reinforcement Learning

5.1 Introduction

5.1.1 Warehousing in e-commerce

Warehousing occupies a central role in a supply chain. According to Gu et al. [60], the primary purposes of a warehouse are to act as a buffer to adapt to the variability of the production flow ; to consolidate products ; and to add marginal value such as pricing, labelling or customisation.

The recent and massive development of e-commerce introduces great challenges in the *Business-to-Consumer* segment. E-commerce involves enormous volumes of orders, and online sales keep growing in number : in 2016, e-commerce sales grew by 23.7%, accounting for 8.7% of the total market [2]. These orders consist, on average, of very few items : the same authors mention that the average number of items per order at Amazon is only 1.6. E-commerce also faces great demand uncertainty, with fast-changing demand trends, and the need to satisfy orders quickly [2–4]. Amazon, for instance, offers customers *Prime*, a differentiated service that used to offer delivery within two days and is now, under certain conditions, promising same-day delivery. Moreover, with easily accessible alternatives available to customers, online retailers face high competitive pressure, and the link between logistics performance and customer loyalty is much stronger compared to other industries [5].

Parts-to-picker Automated Storage and Retrieval Systems (AS/RS), with the typical single, aisle-captive crane retrieving bins from a static rack, have been used for several decades and have proven their superior operational efficiency compared to manual systems, at the cost of an essential initial investment [14]. However, some limitations of AS/RS become apparent when facing new challenges posed by the rise of e-commerce : cranes are sensitive points of failure ; expandability is complicated and expensive ; batching and zoning require consolidation and a delay-sensitive dependency between operators ; and a single crane per aisle can only perform so many cycles. According to Davarzani and Norrman [62], “adaptation is urgent” to keep up with the growing demand.

5.1.2 Robotic Mobile Fulfillment System

Kiva Systems created the first RMFS in 2006, this company was later bought by Amazon and re-branded Amazon Robotics in 2012 [4, 6, 7]. However, RMFSs, also referred to as rack-moving mobile robot based warehouses, or Kiva warehouses, are not limited to Amazon warehouses. Other companies have developed their own systems : for instance, Alibaba’s Zhu Que robots, the Quicktron robots (Huawei), the Open Shuttles by Knapp, CarryPick by Swisslog, Butler by GreyOrange, the Locus Robotics system and others [4, 64].

This new type of automated warehouse involves a large fleet of small robots that move freely around the storage area to retrieve shelves full of items and bring them to operators for picking (possibly after waiting in a queue) before returning to a storage location. Interestingly, when a robot is loaded, it must travel along the aisles, but when unloaded, it can pass under stored shelves to take shortcuts. This type of operations is equivalent to a mini-load with dual-command cycles system with a fleet of non-aisle-captive robots [14]. For a more detailed description of an RMFS and its advantages compared to traditional systems, the interested reader is referred to the following references : [6, 7, 10] and [11].

Rimélé et al. [67] present a mathematical modelling framework that formalises optimisation opportunities in an RMFS. Because e-commerce promises fast delivery of small orders, the fulfilment of requests follows what can be called an *order streaming* logic [12], which "drops orders [...] to the floor as soon as they are received". Because of this specificity, the option of batching requests is minimal, and the system needs to account for uncertainty and adapt online to newly revealed orders. Among the operational decisions, the real-time storage allocation problem can leverage the great storage flexibility of the system. When a robot leaves a picking station, it does not need to store the shelf in its initial location but can choose *any* free location. This allows for the real-time adaptation of the storage layout in response to the demand, thereby minimising the travelling time of the robots, which translates into increasing an upper bound of the throughput.

5.1.3 Contributions and paper structure

This work focuses on a specific operational control aspect of a Robotic Mobile Fulfillment System for e-commerce, namely the problem of learning an effective dynamic storage policy. When a robot is available at an operator's station, a decision must be made in real-time about which storage location to use to minimise the average cycle time of the robots. To make such a decision, one can use information about the next request to process, statistical insights about the demand, and other input regarding the current state of the warehouse's layout. This information at a given time step defines a state's representation in an environment that we approximate as a partially observable Markov decision process (POMDP). We propose the application of a variant of a Q-learning agent [68–70], a Reinforcement Learning (RL) algorithm, to learn an efficient dynamic storage policy that aims at minimising the average cycle time. More precisely, we implement a Differential Double Q-learning agent with experience relays to minimise the average travelling time of the robots. Inspired by *Class-based* storage policies, the decision on where to return a shelf is made based on a discretisation of the storage area into zones. A discrete event simulator is used to evaluate the performance

of our method compared to baselines on different levels of skewed demand distributions. To further improve the performance of our method, we propose an extension using a rollout strategy over a fixed horizon of already-revealed requests. This strategy is also applicable to the baseline methods from the literature.

The presentation of our proposed approach is organised as follows. Section 5.2 first formalises the optimisation problem of dynamically allocating storage locations to shelves, then presents a literature review related to storage policies in automated warehouses and operational control applications specific to RMFS. Section 5.3 presents our methodology, with a discretisation of the storage area into zones, the system representation as a POMDP, and the reinforcement learning variant of a Q-learning agent used to learn an efficient storage policy, as well as the rollout strategy to further improve performance. Section 5.4 explains how we generated data in our simulations and shows results obtained by our method compared with baselines from the literature. Finally, Section 5.5 summarises the obtained results and discusses future research avenues.

5.2 Problem definition and literature

5.2.1 Decision-making framework

In this section, we present the storage allocation decision-making framework, as well as the relevant literature on the topic.

For the case of an e-commerce Amazon-type warehouse, a modelling of operational decisions is proposed by Rimélé et al. [67], with a stochastic dynamic optimisation formulation. In this formulation, an incoming request corresponds to a single item type and consolidation of orders, if any, is assumed to take place in the downstream sections of the warehouse. Orders are revealed online and are considered available for fulfilment as soon as they arrive, with no batching. An *event* corresponds to a robot being available for a task. The tasks of a robot correspond to a sequence of *dual-command (DC) cycles* for which the total time can be decomposed into : (i) storage access time ; (ii) interleaving (or transition) time to retrieval location ; (iii) retrieval time ; and (iv) waiting time at the picking station's queue (see arcs on the left side of Figure 5.1). To create those cycles, storage and retrieval decisions must be made. After the picking operation, the robot is available at the picking station, and it has to perform a *storage task*, which consists in selecting an available location for storage. When the shelf has been stored, the robot needs to perform a *retrieval task*, which consists in choosing which order to fulfil next, from which shelf, and which picking station. Besides those two typical tasks, we also consider an *opportunistic task*, a task that can only occur

in a particular situation where a requested item is stored on a shelf currently being carried by the robot. In this case, the robot can go directly to the back of the waiting line at the operator, without passing by the storage area (right side of Figure 5.1).

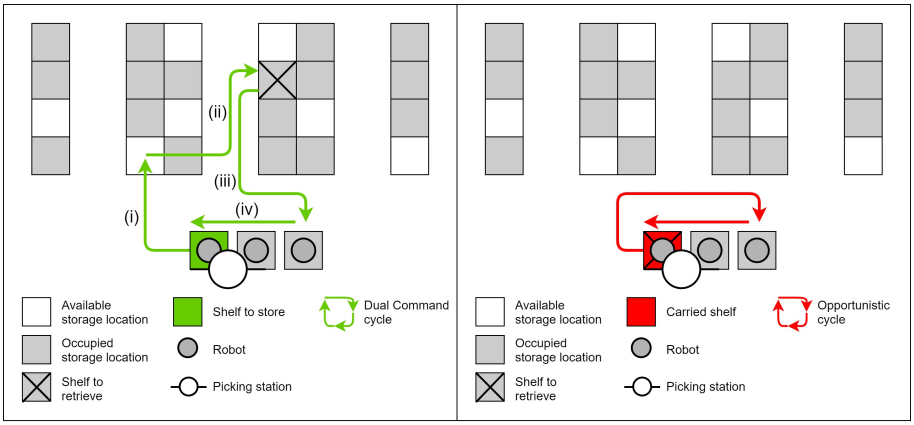

Figure 5.1 Dual-command cycle (left) and opportunistic task (right)

In this work, we focus on optimising storage decisions only. It is assumed that an external system provides the sequence of orders and the decisions on which shelf and picking station to use. To be more precise, this system simply considers orders already revealed and yet to be fulfilled and dynamically sequences them by increasing deadlines, as an emergency rule. If an order requires a shelf carried by another robot, this order is temporarily skipped to be processed by that other robot as an opportunistic task. Also, opportunistic tasks are performed as often as possible. The objective is to minimise the average travelling time, which is directly related to the maximal throughput capacity of the warehouse [71]. We make other simplification assumptions similar to the ones enumerated in the simulation study in [67]. We consider one picking station only, an item is associated with exactly one shelf in the considered storage area, replenishment is explicitly ignored even if some orders could correspond to replenishment tasks, travelling and processing times are deterministic and, finally, congestion of robots in the aisles is ignored.

5.2.2 Storage policies

Commonly used and studied storage policies for AS/RS are *Random* storage policy or policies based on the container's turnover rate. The former gives a container an equal probability to occupy any available location. While this method is space-efficient and easy to implement in practice, it can obviously result in poor accessibility and low productivity during operations. The latter method is typically expressed as a full-turnover-based storage assignment or a

class-based storage assignment. A full-turnover-based assignment locates the most-requested containers closest to the Input/Output (I/O) point ; however, a practical concern appears when new containers enter and leave the system, or when the turnover rates change. In *Class-based* storage, storage locations are clustered into a given number of classes based on their proximity to the I/O point (see left side of Figure 5.2). Then, a container is assigned to a zone based on its turnover rate ; its storage within this zone is random. While this method benefits from both *Random* storage and full-turnover-based assignments, it requires the definition of the number of classes and their dimensions. Numerous analytical and simulation studies such as [20], [21], [23], [22], and [29] show that *Class-based* storage assignment demonstrates the best performance and practicality. However, while those applications generally apply to both single and dual-command cycles system, it seems they have been more driven by the former. In another simulation study, van den Berg and Gademann [27] find out that for dual-command cycles, selecting the storage location that minimises the cumulative distance from the I/O point to the storage location, plus the distance from the storage location to the following retrieval location, gives the best average completion time. This decision rule is often referred to as the *Shortest Leg* storage policy.

5.2.3 RMFS-related work

Some recent work focuses more specifically on RMFS, either on storage decisions or closely related topics like performance measures and others. Lamballais et al. [44] use queueing network models to analytically estimate maximum order throughput, average cycle times, and robot utilisation. Using these models allows one to quickly evaluate different layouts or robot zoning strategies. Among their results, they show the good accuracy of their models compared to simulations. They also show that the throughput is quite insensitive to the length-to-width ratio of the storage area, but is more strongly affected by the location of the picking stations. Also using queueing network models, Roy et al. [47] study the effect of dedicating robots exclusively to either retrieval tasks or replenishment tasks, or a combination of both. They find that the latter reduces the picking time by up to one third, but increases the replenishment time up to three times. They also study the allocation to multiple classes and conclude that allocating robots to least congested classes results in a similar performance as a dedicated zone policy.

Boysen et al. [49] study the problem of synchronising the processing of orders at a picking station with the visits of the shelves carried by robots. By considering a given set of orders, and a capacity to process orders simultaneously, their objective is to minimise the number of visits of shelves to the picking station. They propose a MIP model, a decomposition approach,

and different heuristics and show that they can halve the robot fleet size.

Regarding storage allocation, Weidinger et al. [53] define a *Rack Assignment Problem*, which assigns each stopover (return of a rack) of a given set (batching) to an open storage location. They consider the time at which each rack visits the picking station to be known. They propose a MIP model as a special case of an interval scheduling problem, with the surrogate objective of minimising the total loaded distance. The model formalises the fact that two stopovers can occupy the same storage location only if their storage intervals cannot overlap. They do not consider robots individually, but instead they expect that a robot will always be available for the task. They propose solving their model using an Adaptive Large Matheuristic Search (ALMS), and compare their results with other storage policies, such as those presented in Section 5.2.2. On their surrogate objective, ALMS demonstrates good performance, outperforming the other policies on the total travel time (unloaded robots are considered to travel twice as fast as loaded ones). Similar to the finding of [27], the authors note the very good performance of the *Shortest Leg* policy ; compared to their method, it only increases the travel distance by 3.49% and the size of the robot fleet by 2.17%, without batching.

Yuan et al. [46] propose a fluid model to assess the performance of velocity-based storage policies in terms of travelling time to store and retrieve shelves. They find that when the items are stowed randomly within a shelf, a 2-class or 3-class storage policy reduces the travelling time between 6 to 12%. This theoretical result is validated by simulation. Additionally, they find that if items are stowed together based on their velocity, the travelling distances considered can be reduced by up to 40%.

In situations when batching or zoning requires consolidation, different options can be considered. Boysen et al. [2] consider a manual consolidation with put walls (sorting shelves). They propose a MIP to optimise the release sequence of bins from intermediate storage so that orders are quickly sorted on the put wall and idle times of operators are minimised. In a case where the intermediate storage uses an AS/RS that serves the consolidation area with a conveyor belt, Boysen et al. [72] propose a MIP model for the minimum order spread sequencing problem, which consists in minimising the number of conveyor segments between the first and last occurrence of an order.

Merschformann et al. [13] use a discrete event simulator presented in [58], which considers most, if not all, of the operational decisions occurring in an RMFS. They test a multitude of combinations of decision rules concerning, for instance, the assignment to a picking or replenishment station, the robot selection, storage allocation, etc. They observe significant performance differences between distinct combinations, as well as strong cross-dependencies

between some decision rules, suggesting there may be some benefit to exploring integrated approaches.

5.3 Methodology

This section presents our method for solving the dynamic storage allocation problem in an RMFS. We first define the notion of zones that we use in our storage policy. Then, we explain the representation of the system as a POMDP and the features extraction. In Section 5.3.3, we describe the variant of Q-learning, a Reinforcement Learning agent, that we use. Section 5.3.4 presents implementation details essential to the success of the method. Finally, Section 5.3.5 describes a look-ahead rollout strategy to enhance our storage policy, by making better-informed decisions using already-revealed orders.

5.3.1 Storage zones definition

In this work, inspired by *Class-based* storage policies from the literature, we propose optimising a policy that will allocate shelves to classes instead of exact locations. Contrary to the standard *Class-based* storage assignment presented in Section 5.2.2, the online assignment to a class is not only made based on the turnover rate of the shelf to store, but also on a set of diverse features. Since we are interested in minimising dual-command cycle completion times, as opposed to the access to storage or retrieval time alone (like in single command cycles), the interleaving time between a storage location and the next retrieval location is essential. With the usual definition of classes distributed with an increasing distance from the picking station, such as in Figure 5.2 (left), the access and retrieval times depend directly on the assigned class, but the interleaving time is quite random. For instance, in the ideal case where we store and retrieve shelves from the same storage class 2, we see that the cycle can either be very good (if the two locations are adjacent), or very bad if the locations are at the two extremities. To better account for the importance of interleaving time, we propose another definition of classes, which we now call *zones* as represented in Figure 5.2 (right). These zones are now clustered into regular shapes such that locations within the same zone are close to each other and such that the zones are distributed in an increasing distance from the picking station. With this new zone definition, a storage and a retrieval from the same zone would entail a short interleaving time and, thus, a good cycle. Note that the selection of the exact storage location within a zone is arbitrarily made by choosing the open location closest to the picking station.

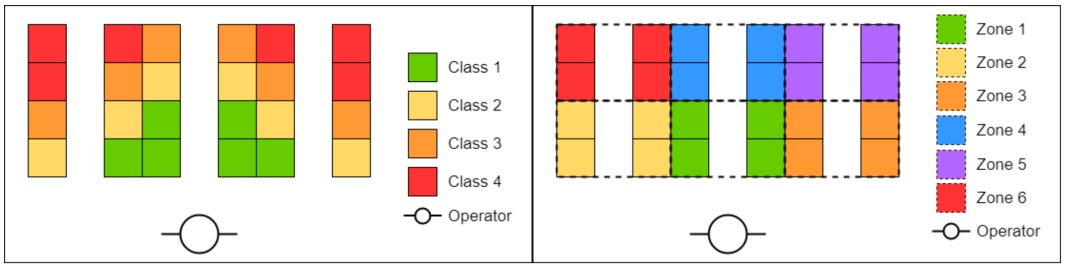

Figure 5.2 Concentric class-based storage layout (left) vs. proposed zones layout (right)

5.3.2 System representation

Rimélé et al. [67] model the storage and retrieval operations in an AS/RS as a Markov Decision Process (MDP), which gives a full representation of the warehouse, including all of the revealed orders, as a state of the system when an event occurs (corresponding to the need for a storage decision here). Instead of giving such a complete representation of the current state of the warehouse, we propose a *Partially Observable* MDP (POMDP), where only some characteristics of the real state are used as state representation. In the traditional *Class-based* storage policy, for instance, the representation of the state is simply the turnover rate of the shelf to store, and based on this information the robot knows which class to assign it to.

We decided to represent a state with some more characteristics (features), given below :

- Average turnover rate of the shelf to store
- Relative rank of the shelf's turnover rate
- Zone of the next retrieval task (one-hot encoding, one feature per zone)
- Occupation levels of each zone (one feature per zone)
- For each zone, the number of robots that are currently moving toward the zone to retrieve a shelf

An extra feature is used to encode whether the next task is an opportunistic one (the shelf is not stored but is already being carried by the robot). In total, if n denotes the number of zones, there are $3n + 3$ scalars in the features vector.

The intuitive idea is that there must be a balance between greedily minimising the immediate cycle time, similar to the *Shortest Leg* storage policy, and minimising the future access time, similar to a turnover-based storage policy as presented in Section 5.2.2. The objective is to learn a policy that, based on the above set of features, will aim at minimising the average cycle time.

5.3.3 Reinforcement learning method

We propose using a Reinforcement Learning (RL) method to learn an efficient storage policy. The field of RL offers a diversity of methods to solve decision problems whose objectives are defined recursively by a Bellman equation, introduced in dynamic programming (DP). Compared to traditional DP methods that require complete information about the model and probability distributions, RL methods can learn from repeated experiences, even in large state and action space, when using function approximations. As presented by Sutton and Barto [73], in RL the decision-maker, or agent, interacts with its environment. At every time step t, and based on the current representation of the state S_t, the agent selects an action A_t, which impacts the environment and receives from it a reward R_{t+1} and a new state representation S_{t+1}. Based on repeated experiences, the goal of the agent is to learn a policy that maximises the onward cumulative sum of rewards. Note that the agent is not necessarily given the complete state characteristics ; it can only be given a partial representation through a POMDP. The state representation defined in Section 5.3.2 is the partially observable state given to the agent in our approach. In value-based RL, given a policy π, the agent learns the State-Action value function $Q_\pi(s,a) = \mathbb{E}\left[R_1 + R_2 + ... \mid S_0 = s, A_0 = a, \pi\right]$ that represents the expected (possibly discounted) cumulative reward at state s if action a is taken, if policy π is followed afterwards. The optimal value function Q^* is such that $Q^*(s,a) = \max_\pi Q_\pi(s,a)$ for all state and action pairs, and obtaining a decision policy from a value function simply consists in acting greedily with regard to action values : $A_\pi(s) = \text{argmax}_a Q_\pi(s,a)$.

In Q-learning [68], the value function is iteratively updated by sampling based on the Bellman equation : $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$, so the update uses the received reward and the current estimate of the next state value (bootstrap) as its target. Importantly, for convergence to optimality, the Q-learning agent needs to correctly learn the values of its actions (exploitation) while trying new actions, possibly more promising (exploration). This trade-off is central to any RL method, and here we will consider an ϵ-greedy selection rule for this purpose (ϵ % of the time, the greedy action is selected, otherwise a random action is taken).

Even if our state space has a small number of dimensions and the number of actions is also small, calculating the value function individually for every state-action pair with a tabular method would be out of reach considering the exponential number of combinations. Instead, Deep Q-learning uses a neural network (NN) as a function approximator that takes the state features as inputs and outputs the estimated values for each possible action [70]. Using function approximators allows to escape the curse of dimensionality by leveraging information about similar states and by generalising to unseen state-action pairs using only a limited

number of parameters. Let us denote by $\mathbf{w_t}$ the current parameters, weights, of the neural network function approximator and by $Q(s, a, \mathbf{w_t})$ the value of state s - action a estimated by this neural network. Because a neural network is differentiable, we can minimise the square loss error between the current estimate value and the target by taking a step in the opposite direction of the value function's gradient : $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a, \mathbf{w_t}) - Q(S_t, A_t, \mathbf{w_t})]\nabla Q(S_t, A_t, \mathbf{w_t})$.

In the case of an infinite time horizon system, such as ours where retrieval tasks never end, rewards cannot simply be accumulated because the state values would never converge but rather would diverge to infinity. The most common solution is to apply a discount factor γ in the Bellman equations, such that for a given policy π we have : $Q_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma\, Q_\pi(S_{t+1}) \mid S_t = s]$, with $0 < \gamma < 1$. The justification for using a discount factor is first for the mathematical convenience of convergence of the cumulative finite rewards. Also, in some systems, a discount factor can represent an economical discount rate, or at least the idea that rewards received now are worth more than they would be later. Yet in our case, no incentive should be given for early performance since operations throughout the day have equal importance and no termination occurs. An alternative to discounted rewards is what is called *differential returns* or *average rewards* [73]. Introduced in Dynamic Programming [74] and later in Reinforcement Learning (R-Learning in [68]), the idea is to optimise the average sum of reward $\lim_{n\to\infty} \frac{\sum_{t=1}^{n} R_t}{n}$ by considering return values equal to $R_t - \bar{R}$, with $\bar{R}$ the average reward of policy π. This setting has the interesting property of having the sum of returns converging (to 0) while maximising the average reward in the process. When action A_t of policy π is taken, the state-action value update rule becomes $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R - \bar{R} + \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$.

Q-learning is known to suffer from an overestimation of actions' values. Two main reasons explain this phenomenon. First, in its update mechanism, the maximum value of the next state-action pair is used as an approximation for the maximum expected value. Second, both the action selection and value estimation come from the same maximum operator that suffers from estimation errors, and thus favours overestimated values. To answer this problem, van Hasselt [75] proposes a Double Q-learning method in the tabular case, which was later extended to Deep Q-learning in [76]. For Deep Q-learning, the general idea is to separate the action selection from the value estimation using two neural networks, the online NN and the target NN. The authors demonstrate that this approach eliminates the positive bias on the action values and results, in general, in better performance. The online NN is updated as before, but the target NN is not. Instead, it receives a copy of the weights of the online NN every K iterations.

Algorithm 5.1 combines the different elements presented above into one agent, denoted as *Double Differential Deep Q-Learning*. Step 1 initialises a starting state, the average reward (arbitrarily set, for instance, to 0) and the weights of both the online and target neural networks, equal to each other. For every iteration, Step 3 selects an action, greedily or not, takes it, and observes the reward and the next state. Step 4 calculates the target value for state S_t - action A_t pair. Step 5 does the gradient descent to update the online network. If the action selected A_t was greedy with respect to the online network, Steps 6 to 8 update the average reward $\bar{R}$. Finally, every K iterations, the target network receives a copy of the online network weights (Steps 9 to 11).

Algorithm 5.1 Double Differential Deep Q-Learning

1: initialise state $S_0, \bar{R}, \mathbf{w_t} = \mathbf{w_t^-}$
2: **for** $t = 0$ to $+\infty$ **do**
3: take action A_t (ϵ-greedy wrt $Q(S_t, ., \mathbf{w_t})$), observe R_{t+1}, S_{t+1}
4: $Y_t \leftarrow R_{t+1} - \bar{R} + Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a, \mathbf{w_t}), \mathbf{w_t^-})$
5: $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} + \alpha[Y_t - Q(S_t, A_t, \mathbf{w_t})]\nabla Q(S_t, A_t, \mathbf{w_t})$
6: **if** $A_t == \underset{a}{\text{argmax}}\, Q(S_t, a, \mathbf{w_t})$ **then**
7: $\bar{R} \leftarrow \bar{R} + \beta[Y_t - Q(S_t, A_t, \mathbf{w_t})]$
8: **if** $t \bmod K == 0$ **then**
9: $\mathbf{w_t^-} \leftarrow \mathbf{w_t}$

5.3.4 Implementation details

In our experiments, we use a fully connected feed-forward neural network as a function approximator. While the neural network has one output per storage zone, which represents the expected cumulative cycle times onwards (relative to the average cycle time), if the storage zone is selected, some zones may not be selected if they are currently full. For this reason, when we select an action based on its value out of the neural network, we first need to filter out full zones.

Another point worth mentioning deals with backpropagation when an opportunistic task is performed. Such a task is enforced when possible, which occurs when the next retrieval zone is not a zone but the picking station (encoded in the features extraction). Even though the agent does not require any action to be taken, it is essential to backpropagate the observed reward in the neural network for all of the actions. The reason for that is bootstrapping. If a non-opportunistic task transitions to a state that will enforce an opportunistic task, the state-action value update in Q-learning uses the observed cycle time as well as the estimated value of the future state, which, in this case, requires an opportunistic task. The neural

network must then be accurate for such a state and learn that the values of all the storage zones are to be equal to the value of an opportunistic task.

As in [70], we use experience replay. It consists in storing simulation experiences (vector $S_t, A_t, R_{t+1}, S_{t+1}$) into a fixed size replay memory. Instead of training after each observed experience, we wait for some iterations before sampling a batch of experiences from the replay memory and then train the neural networks with mini-batches. This batch training affords more training stability, allows the reuse of past experiences, and speeds up the training step.

Finally, to make better use of available information, particularly requests that are already revealed at a given time step, we propose an extension to using only Q-values. A look-ahead rollout strategy is proposed. We present this method in Section 5.3.5 and the results of our Q-learning application with and without rollouts is presented in Section 5.4.

5.3.5 Look-ahead rollout strategy

Monte Carlo Tree Search (MCTS) methods are exploration techniques to improve policies, based on previous work on Monte Carlo methods. Coulom [77] first describes a tree search method applied to games. Several variants have been adapted for numerous applications, such as the famous AlphaGo Zero from [78]. The general idea behind MCTS is to explore the action space by simulations following a tree structure, further exploring the promising areas of the tree.

For the RMFS real-time storage assignment, the objective of a look-ahead search is twofold. First, as in other applications, it helps the agent take better actions by exploring numerous possible scenarios. Second, and more specific to our application, the look-ahead can leverage information that is already revealed but not necessarily given in the state representation. This is the case of future orders. So far, in Section 5.3.2, only information about the next order the robot needs to retrieve is included in the state features. However, at a given time step, while not all future orders have been revealed, some have. The look-ahead can then use *only* those already-revealed orders (even if the real future sequence will differ) to take a better informed first action.

Figure 5.3 illustrates this point. The orders o_i are sorted according to the sequence in which they will actually be processed, but at time t_0 only the *green* orders $\{o_1,\ o_2,\ o_4,\ o_5\}$ are known. The *orange* orders $\{o_3,\ o_6,\ o_8\}$ will be revealed later at time $t_1 > t_0$, for instance by the time order o_2 will have been processed. In this case, order o_3 is be inserted before o_4, for priority reasons. In this case, even if new orders will later be inserted into the list, at time

t_0 only requests $\{o_1,\ o_2,\ o_4,\ o_5\}$ are used in the rollout, the only exceptions being potential other *green* requests not depicted on Figure 5.3.

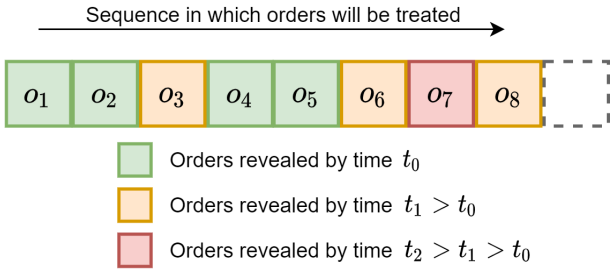

Figure 5.3 Revealed orders at decision time

Bertsekas [79] presents different approaches for MCTS implementation. One of these approaches is a single-step look-ahead with truncated rollout and cost approximation. Figure 5.4 presents the implementation of this type of rollout adapted to our problem. At first, all feasible actions at a given state S_0 (time t_0) are considered and their values $V(S_0, A_i)$ and visit counts n_{A_i} are set to 0. At each iteration, an action is randomly selected (action A_1 in Figure 5.4) and a simulation of h (horizon) consecutive storage tasks is run. This finite simulation uses orders revealed by time t_0. The action selection in the simulation uses the agent's current policy. The sum of reward along the simulation is computed, and the value of the last state S_h is, by bootstrapping, set as the Q-value of the agent. Then, the value of the selected action is updated to its experienced average, and the visit counter is incremented. The process is repeated for a given number of iterations, and finally, the action presenting the best (min) value is selected. Note that in our specific usage of this look-ahead rollout strategy, each feasible action needs to be selected only once. Indeed, both the policy and the transitions within the rollout are deterministic because of the Q-learning agent's policy and because we do not simulate unknown future orders.

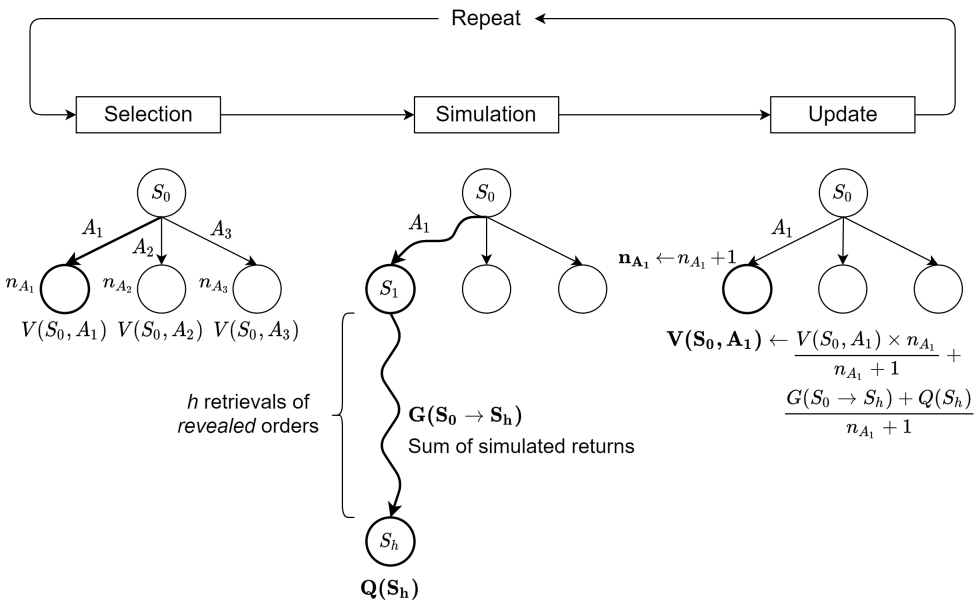

Figure 5.4 Look-ahead rollout strategy

5.4 Simulation study

In this section, we present a simulation study comparing our proposed methods to typical storage policies found in the literature. We start by defining the framework of the study, before presenting our results.

5.4.1 Parameters

Kiva and Amazon have claimed travel speeds up to 3 to 4 miles per hour ; however, to account for turns and congestion, we assume that the robots travel at a constant speed of 1.4 miles per hour (about 0.6 m/s). In our experiments, we assume an average picking time by the human operators of 8 sec and a loading and unloading time by the robots of 3 sec. Our study considers a relatively small storage area, with 36 storage locations (6 zones of size 2 by 3 shelves) for 34 shelves, 5 automated robots and one picking station.

Incoming orders in warehousing systems are typically skewed, with a small percentage of the items representing the vast majority of the orders and the others qualified as slow movers. Hausman et al. [20] propose modelling such a demand distribution with the definition of an ABC curve : $G(x) = x^s \quad \text{for } 0 < s \leq 1$, which represents the proportion of cumulative demand (%) of the first x% of the shelves. The smaller the skewness parameter s, the more skewed the demand distribution. We follow this suggestion by associating with every item type

i (equivalent to the corresponding shelf), at every discretised period, a Poisson distribution of average $\lambda_i = \left(\left(\frac{i}{m}\right)^s - \left(\frac{i-1}{m}\right)^s\right) \times \frac{n}{N}$, where m is the number of item types, n the total number of orders in the time horizon, and N the number of discretised periods. Then, for every generated order k, a completion time δ_k is randomly drawn from a uniform distribution of interval $[1; \alpha \times T]$ where α defines the tightness of the completion times. Note that in the case where a drawn demand for a given item type is greater than one, the resulting orders are automatically grouped, corresponding to an opportunistic task of type 2 in [67]. The values used here are : $m = 34$, $n = 30000$, $N = 1440$ (corresponding to discretised periods every 1 min) for a time horizon of 24h, and $\alpha = 0.4$. The values of the skewness parameters will be varied to study their impact on the different storage policies.

Figure 5.5 Simulation study - Plan view of the storage area

After some trial and error, we set the parameter values for the RL agent as follows. Each training phase is conducted over 10000 episodes, corresponding to 24h of simulated incoming orders. Each of these training episodes uses its own generated instance of demand, while the comparisons between the baselines and the test of the RL policy are made on another shared instance. The exploration rate ϵ is kept constant at 0.1. The neural networks are created using the Keras library. We use the mean square error loss function and the Adam optimiser for gradient descent. Both neural networks are feed-forward, fully connected with 3 inner layers of 32 neurons each and Relu activation functions. The output layer presents one neuron per zone of storage (6), and these neurons are linearly activated. The learning rate is set to 0.00025, the replay memory has a capacity of 1000, and batch training is performed every 100 iterations. This batch consists of 256 experiences sampled from the replay memory, and they are fitted to the model in mini-batches of 64. The target network's weights are updated

every 500 iterations.

5.4.2 Results

This section presents results obtained by the proposed approach and the standard baselines. First, experiments were run using the method presented in Section 5.3.3 on different instances corresponding to different values of demand skewness parameter s. Then, the rollout strategy presented in Section 5.3.5 was applied on both the Q-learning agent and the best-performing baseline, and gains that can be obtained by using additional available information were observed. The three baselines tested were the following : *Random* storage policy, *Class-based* and *Shortest Leg*.

The *Class-based* storage policy uses the concentric classes definition, depicted in Figure 5.2 (left) and conventionally described in the literature. Since our method allocates shelves to zones instead of exact locations, the *Shortest Leg* storage policy is implemented similarly. When a storage task needs to be performed and the location of the next retrieval is known, the exact potential location within each zone can be identified because of the arbitrary choice of selecting the location closest to the picking station. Knowing the candidate location from each zone (if any), the *leg* distance can be computed as being the distance from the picking station to the storage location plus the distance from the storage location to the next retrieval. The selected zone corresponds to the smallest leg value. This policy results in greedily minimising the immediate cycle times.

Table 5.1 presents the average travelling times, $t(s)$, obtained from the three baselines and the RL agent, without rollouts, for several skewness parameter values ranging from $s = 0.4$ (very skewed) to $s = 1$ (flat distribution). For each storage policy, the corresponding performance gain, g(%), compared to the *Random* storage policy is also computed ; this represents the percentage decrease in travelling time. As expected, the performance of the *Class-based* policy increases when the skewness parameter value s decreases. In concordance with the literature, we observe the good performance of the *Shortest Leg* policy, which remains rather constant, between 10 and 11% of improvement, depending on the skewness parameter values. In fact, it is only surpassed once by the *Class-based* policy for the most skewed distribution $s = 0.4$, but it offers the net advantage of being independent of the distribution pattern. The Q-learning agent consistently performs better than the baselines, between 3.21% and 4.83% of additional gain compared to the baselines relative to *Random* storage. Similar to the SL policy, its performance remains rather independent of the skewness of the distribution. Figure 5.6 presents a typical training curve, here for parameter $s = 0.6$. The x axis shows the number of training episodes ranging from 0 to 10000 and y corresponds to the relative

gain in travelling time compared to the *Random* storage policy. The horizontal dashed lines in the graph represent the performance of the three baseline policies. The test policy curve represents the performance of the RL policy in training, which is tested every 100 training episodes. With our training settings, we notice the non-linear evolution of the test policy curve until it reaches some plateau value around 5000 episodes.

Table 5.1 Average travelling times t(s) and performance gains g(%) depending on distribution skewness parameter value s - without rollouts

Storage policy	Random		Class-based		SL		Agent	
	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)
s=0.4	37.97	-	33.41	12.01	33.61	11.48	32.19	15.22
s=0.5	38.62	-	34.65	10.27	34.18	11.48	32.90	14.79
s=0.6	38.69	-	35.34	8.65	34.79	10.09	32.92	14.92
s=0.7	38.73	-	36.01	7.02	34.70	10.39	32.91	15.02
s=0.8	38.94	-	36.48	6.30	34.76	10.73	33.04	15.15
s=0.9	38.98	-	37.41	4.03	34.53	11.42	33.03	15.27
s=1.0	39.06	-	38.26	2.05	34.76	11.01	33.4	14.48

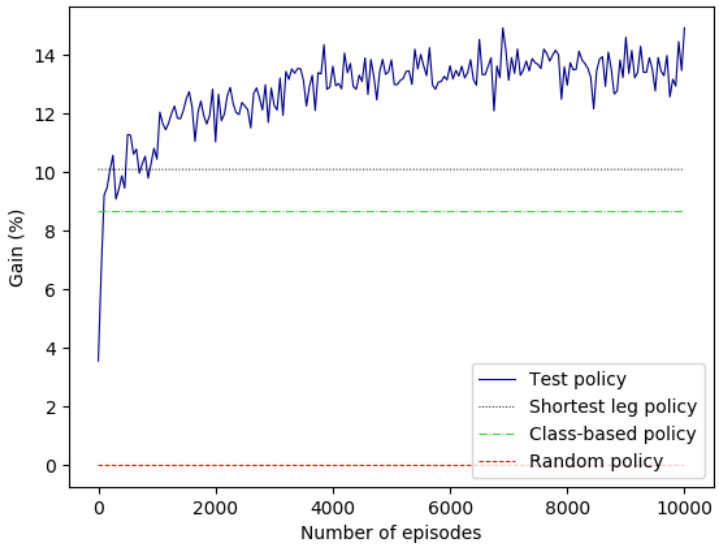

Figure 5.6 Typical instance of a training curve (here for s=0.6)

Over a second phase, we retained the overall best performing-baseline (SL) and our Q-learning agent to enhance them using the look-ahead rollout strategy presented in Section

5.3.5. Again, we tested the resulting policies on different skewness parameter s values, but we also considered different horizon values h, which define how far ahead the rollout is run. The only difference in the application of the rollout strategy to the SL baseline compared to the Q-learning agent is the absence of a Q-value at the last step of the rollout. In this case, the SL policy is run over the next h already-revealed orders, and the average of the resulting cycle times is computed, for each first feasible action. Table 5.2 presents the obtained results in terms of performance gains (%) compared to *Random* storage. First, we notice that for all parameter values, the performance is significantly improved. We can see that for both policies, the performance varies depending on the horizon of the rollout that is considered. As a general statement, it appears that horizon values 20 and 30 give the best results, with some exceptions. The fact that the performance increases with the horizon value before decreasing again is not too surprising. While using revealed orders in the rollouts is beneficial to making better-informed decisions, the longer the rollout, the more likely it is that new, unknown orders will be later inserted between the orders considered in the rollout. When this phenomenon starts to occur too often, it deteriorates the insight of the rollout and results in worse policies. The values in bold in the table designate the best results for each parameter s for both policies. For the SL storage policy, the application of the rollout strategy increases its performance gains between 5.26% ($s = 1.0$) and 7.11% ($s = 0.7$). For the Q-learning agent, the performance gains range between 2.78% ($s = 0.8$) and 4.54% ($s = 0.4$). It appears that using rollouts benefits strongly skewed distributions slightly more than less skewed ones, which is most likely explained by more opportunities associated with fast and slow-moving shelves. Also, the rollout strategy benefits the SL policy more so than the agent. The differences in performance gains between the two policies with rollouts are now 2.29, 0.83, 1.47, 1.07, 1.55, 2.2 and 1.29% for increasing s values. Using rollouts gives a policy insights about future behaviours by simulating possible trajectories. While the Q-learning agent still benefits from this insight, with the Bellman equation it is, by design, already taking into account future impacts of a decision. On the other hand, the SL policy acts perfectly greedily ; it has more to gain by looking into possible future outcomes. While the combination of the Q-learning agent with rollouts still performs best with an average of 1.53% of performance gain compared to SL with rollouts, this assessment has interesting practical value because of the relative simplicity of implementation of SL with rollouts and its highly competitive performance. Finally, when comparing the results of the complete method with the best baseline commonly used from the literature (SL without rollouts), an average performance gain of 7.58% is found.

Table 5.2 Performance gains g(%) depending on distribution skewness parameter value s and horizon h - with look-ahead rollouts

	SL + rollouts					Agent + rollouts				
h value	5	10	20	30	40	5	10	20	30	40
s=0.4	14.01	16.83	**17.47**	17.07	17.22	16.26	17.01	**19.76**	18.84	18.80
s=0.5	14.91	16.03	17.57	**17.65**	16.00	15.17	17.14	**18.48**	18.41	17.89
s=0.6	14.97	15.47	16.06	**16.92**	16.05	15.71	17.45	**18.39**	17.99	18.16
s=0.7	14.41	15.45	**17.50**	16.22	15.85	14.96	16.82	18.13	**18.57**	18.11
s=0.8	14.23	15.51	**16.38**	15.42	15.46	15.13	16.32	17.23	**17.93**	16.96
s=0.9	14.22	15.49	16.34	**16.79**	16.52	15.11	17.04	17.83	18.95	**18.99**
s=1.0	15.15	15.60	**16.27**	15.64	15.61	15.27	16.81	17.34	**17.56**	17.17

5.5 Conclusions

This work tackles the problem of dynamically allocating shelves to storage locations within a Robotic Mobile Fulfillment System. Because of the very nature of the e-commerce market, new orders need to be considered, if not fulfilled, as soon as they are revealed, which limits opportunities for batching. Typical methods from the literature correspond to decision rules which rely either on the distinct turnover rates of the containers or on minimising immediate cycles greedily. After making several assumptions about other operational decision rules in such warehouses, as well as the physical characteristics of the warehouse, we propose defining a storage policy using a POMDP and a Q-learning agent from Reinforcement Learning, to minimise the average travelling cycle time. This agent learns from repeated experiences which storage decision should be made based on a set of features representing the current state of the warehouse. Using zone-based storage allocation, we compared our method to decision rule baselines, including the *Class-based* storage policy and *Shortest Leg* storage policy. A performance gain between 3.5% and 5% higher than the best baseline relative to *Random* storage was observed. In a second phase, the objective was to leverage additional information regarding orders that are already revealed at a given time step. While new orders will later appear and be inserted between those revealed orders, the latter can still be used in a look-ahead rollout strategy applied at the decision-making step. This rollout simulates finite horizon trajectories using the storage policy at hand for action selection within the trajectory, as well as a Q-value estimation at the last step, acting as an estimation of future expected objective value. Such a rollout strategy has the benefit of being applicable to any storage policy, which allowed us to make fair comparisons between the Q-learning agent and the *Shortest Leg* policy, both using rollouts. After selecting the proper horizon length, using look-ahead rollouts increased the performance gains of the SL and Q-learning policies

by about 6 and 4%, respectively. Even though the Q-learning with rollouts policy gave the best results, the higher positive impact of rollouts on the SL policy makes it an interesting contender, due to its relative simplicity of application. Comparing the policy obtained from Q-learning with rollouts to the best baseline, we observed an average performance gain of 7.5%, which, we think, demonstrates the potential for more research in this direction.

Future research may build on this work to extend it further. First, storage allocation could select exact open locations instead of using the notion of zones as described here. Instead of using single-step look-ahead rollouts, one could implement a complete Monte Carlo Tree Search algorithm to better explore action selection. The expensive computational cost of such an approach may limit online deployment but, coupled with a policy gradient agent instead of a Q-learning one, the tree search could be used extensively in the training phase only for fast future deployment. Another aspect worth looking into would be the waiting time of robots at the picking stations. In this work, we only consider the travelling time, which is a lower bound to the full cycle time. Aiming at minimising the travelling time along with implementing a distributed arrival of robots at the picking stations could result in higher throughput, but how to do that exactly appears to be quite challenging.

CHAPITRE 6 ARTICLE 3 : SUPERVISED LEARNING AND TREE SEARCH FOR REAL-TIME STORAGE ALLOCATION IN ROBOTIC MOBILE FULFILLMENT SYSTEMS

Adrien Rimélé[ab], Philippe Grangier[d], Michel Gamache[ac], Michel Gendreau[ab], Louis-Martin Rousseau[ab]

[a]Department of Mathematics and Industrial Engineering, Polytechnique Montréal, [b]CIRRELT, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, [c]GERAD, Group for Research in Decision Analysis, [d]IVADO Labs

Cet article a été soumis à *INFORMS Journal on Optimization* le 31 mai 2021.

ABSTRACT

A Robotic Mobile Fulfillment System is a robotised parts-to-picker system that is particularly well-suited for e-commerce warehousing. One distinguishing feature of this type of warehouse is its high storage modularity. Numerous robots are moving shelves simultaneously, and the shelves can be returned to any open location after the picking operation is completed. This work focuses on the real-time storage allocation problem to minimise the travel time of the robots. An efficient – but computationally costly – Monte Carlo Tree Search method is used offline to generate high-quality experience. This experience can be learned by a neural network with a proper coordinates-based features representation. The obtained neural network is used as an action predictor in several new storage policies, either as-is or in rollout and supervised tree search strategies. Resulting performance levels depend on the computing time available at a decision step and are consistently better compared to real-time decision rules from the literature.

KEYWORDS

Real-time decision-making ; E-commerce ; Storage Policy ; MCTS ; Supervised learning

6.1 Introduction

E-commerce has not only revolutionised access to goods, it has fundamentally changed the entire Business-to-Consumer market segment. Its growth in the last decade has been steady, with an annual increase of approximately 15% in the US market. The year 2020, exceptional in all regards, reached new summits with a stunning 44% increase in US e-commerce sales [1].

E-commerce market share now represents 21.3% of all retail shares, totalling $861B in the US alone. Specific features of online sales, as well as fierce competition between retailers, put intense pressure on the supply chain, including the warehousing systems.

In response to increasing demand, new warehousing systems have appeared in recent years. One of them is the Robotic Mobile Fulfillment System (RMFS), also referred to as a rack-moving mobile robot based warehouse, semi-automated storage system, or Kiva warehouse [4, 6]. In this parts-to-picker warehouse, a fleet of small robots is tasked with retrieving storage shelves and bringing them to picking stations where human operators pick items to fulfil orders. Kiva Systems first introduced the RMFS in 2006 before being bought by Amazon in 2012 and re-branded as Amazon Robotics. Similar systems have been introduced by other companies, including the Zhu Que robots (Alibaba), the Quicktron robots (Huawei), the CarryPick system (Swisslog), and others [4, 12]. Sales of warehouse robots are expected to reach $30.8B by 2022 for around 900,000 robot units sold per year [9]. In an RMFS, the items inventory is stored in mobile shelves that robots can lift and bring to a picking station. The robots typically perform *dual command cycles*, which consist of immediate successions of storage and retrieval tasks. Interestingly, the robots have to move along the aisles when loaded but can pass under the stored shelves when unloaded to avoid traffic. For more technical information about RMFSs, the interested reader is referred to [6, 7] and [4]. A distinguishing feature of an RMFS is its high storage modularity, since the robots operate across the entire storage area simultaneously and can select any free storage location when returning a shelf. This presents great opportunities to continuously adapt the layout of the warehouse in order to accommodate new incoming orders.

This work focuses on the real-time storage allocation problem to minimise the average travel time of the robots, which is inversely proportional to the maximum order throughput value. In this problem, stochastic incoming orders are continuously revealed in the list of orders to fulfil, and a storage location must be chosen as soon as a robot is released by an operator at a picking station. The contribution of this work is to propose a new approach based on supervised learning to this real-time problem. First, an efficient – yet computationally expensive – Monte Carlo Tree Search (MCTS) method is proposed to generate high-quality experience offline. This experience is later used to learn a policy that performs better than baselines from the literature while being just as fast to deploy. The features representation, using the shelves' spatial coordinates in the storage area, is crucial to successful supervised learning by a neural network. With additional computing time at the decision step, the learned policy is further improved by a rollout strategy and a newly introduced supervised tree search method, which provide significant performance gains.

The remainder of this paper is organised as follows. Section 6.2 presents a literature review on RMFSs. Section 6.3 describes the problem and the different assumptions that are made. Section 6.4 explains the methodology, starting from the offline MCTS method, followed by the supervised learning of a storage policy, and enhancements with rollout strategy and a new supervised tree search method. Section 6.5 presents a simulation study of the proposed methods compared to baselines from the literature. Finally, Section 6.6 draws conclusions and discusses future research perspectives.

6.2 Literature review

While the deployment of RMFSs is relatively recent, research on the topic has its origins in the field of Automated Storage and Retrieval Systems (AS/RSs) starting in the 1970s [14, 80]. In its most typical implementation, an AS/RS corresponds to a juxtaposition of static racks, each with a single, aisle-captive crane retrieving bins. Just as in RMFSs, an AS/RS's crane performs dual command cycles. The fact that RMFSs use a fleet of robots that can simultaneously access the entire storage area significantly changes the operational decision-making problems. Due to the unique challenges of RMFSs, specific literature has developed on the topic in the last few years.

6.2.1 Analytical models

Several papers develop analytical models to quickly estimate key performance indicators. Lamballais et al. [44] propose semi-open queueing network models to estimate the maximum order throughput, average order cycle time, and robot utilisation, depending on layout decisions and robot zoning strategies. They demonstrate the accuracy of their model compared to results from simulations, and note that the throughput is more strongly influenced by the location of the picking stations than by the dimensions of the storage area. Zou et al. [45] present a semi-open queueing network to evaluate decision rules to assign robots to picking stations, and show that their rule performs better than a random assignment. In [46], the authors propose a fluid model to assess the performance of velocity-based storage policies in terms of travelling time to store and retrieve shelves. They find that when the items are stowed randomly within a shelf, a 2-class or 3-class storage policy reduces the travelling time anywhere from 6 to 12% (a class regroups containers by demand rate, and classes of higher demand are typically localised closer to the picking stations). Additionally, they find that if items are stowed together based on their demand velocity, the loaded travelling times can be reduced by up to 40%. Roy et al. [47] use a closed queueing network model to evaluate the impact of allocating robots exclusively to either retrieval or replenishment tasks, or a combi-

nation of both. They conclude that the third option reduces picking time by 30% but increases replenishment time by a factor of three. They also study storage allocation to multiple classes and find that sending robots to the least-congested class gives similar performance compared to a dedicated class policy. Lamballais et al. [48] propose a semi-open queueing model to optimise the number of shelves per item type, the ratio of the number of picking stations to replenishment stations, and the replenishment level per shelf. They conclude that there is a significant throughput gain when : the same item is stored in multiple shelves, the ratio of picking stations to replenishment stations is optimised, and a shelf is replenished before it is empty.

6.2.2 Scheduling and task allocation

Other papers focus on scheduling the processing of orders at the picking stations. Boysen et al. [49] present a Mixed Integer Program (MIP) to minimise the number of shelf visits by synchronising the orders processing and shelves' arrival at a picking station. They propose several heuristic methods based on dynamic programming and simulated annealing to solve the model. Through simulations, they show that their approach can halve the robot fleet size in the best case. Gharehgozli and Zaerpour [50] address the problem of scheduling a batch of orders for a single robot to minimise the travelling time. They consider two cases where a shelf either must be stored back at its exact same storage location or can be returned to a set of locations. They demonstrate the efficiency of their adaptive large neighbourhood search method and show that the scheduling improves travelling time by 24% compared to a random order. Valle and Beasley [9] present a mathematical model and matheuristic solving methods to simultaneously allocate a batch of orders and shelves to multiple pickers. They also propose a model to sequence the shelves' visits to the picking stations. Bolu and Korcak [51] propose a task planning approach that aims at minimising the number of shelves needed to fulfil a batch of orders. Then, they dynamically adapt the system by selecting tasks for the picking stations and the robots. Through an extensive simulation study, they demonstrate their method's capacity to significantly reduce the completion time and maintain a high pile-on (i.e., the number of items retrieved from a single shelf visit).

From a different perspective, Zhang et al. [56] define the problem of allocating robots to picking tasks as scheduled at the picking stations. They formulate a resource-constrained scheduling problem and propose a solving approach using a serial schedule generator and a genetic algorithm. They show that their method obtains better results than commonly used rule-based scheduling methods. Xie et al. [57] propose an MIP to assign shelves to stations and orders to stations simultaneously. They also consider splitting multi-line orders for

picking at different stations. They use the discrete event simulator RAWSim-O from [58], which realistically reproduces operations in an RMFS, to compare their results with a sequential allocation. With split orders, they obtain a 46% increase in throughput. Using this simulation framework, Merschformann et al. [13] assess combinations of decision rules for operational problems such as picking and replenishment station assignment, pod selection, and storage assignment. In particular, they note the importance of careful selection for each application, the significant influence of the picking station assignment rule, and the high cross-dependencies between some rules. Rimélé et al. [67] present a stochastic mathematical modelling framework that formalises real-time decision-making. Presented as a stochastic dynamic program, it considers most operational decisions from order sequencing to storage allocation, with stochastic incoming orders. Its main objective is to set a framework to stimulate the development of advanced methods for real-time decisions in RMFSs.

6.2.3 Storage allocation

Regarding storage allocation specifically, Krenzler et al. [52] present a deterministic MIP model aiming to minimise the shelves' storage and retrieval times. They consider that the shelves' visit times at the picking station are known and assume that the waiting queues are always full, which frees up more storage locations. They present a variety of solving methods and demonstrate their effectiveness compared to a random storage policy. Weidinger et al. [53] present an MIP model in the form of an interval scheduling problem to minimise the loaded travel time. They also consider that the visit time of each shelf at the picking station is known. With this assumption, the model enforces that two storage allocations of shelves can occupy the same location only if the corresponding storage intervals do not overlap. They present an Adaptive Large Matheuristic Search solving method and obtain conclusive results in terms of loaded and complete travel times, compared to standard decision rules. They note the good performance and practical aspect of the Shortest Leg policy (referred to as Shortest Path), which only increases the travel time by 3.49% and does not require any batching of orders. Mirzaei et al. [54] use product affinity (i.e., the probability of products being ordered together) and turnover rate to jointly assign products to shelves, and shelves to locations, to minimise the retrieval time with an integer model. They show that their model reduces retrieval time by up to 40% compared to traditional full-turnover and class-based storage policies. Rimélé et al. [81] use Reinforcement Learning techniques to learn a zone-based storage policy that minimises the travel time of robots with one picking station. Depending on the distribution of orders, they improve the Shortest Leg (SL) policy's performance by 3.5 to 5%. They also propose a lookahead rollout strategy that uses already-revealed orders to simulate the near horizon. Compared to SL, this approach generates results that are 7.5%

better on average.

This paper is presented as a continuation of [81], as its objective is to learn an efficient real-time storage policy. However, the methods proposed here select exact storage locations instead of zones and rely on a significantly different learning approach. The following section defines the problem we consider and the underlying assumptions.

6.3 Problem definition

This work focuses specifically on storage allocation when a robot has to return a shelf to the storage area, with the objective of minimising the average travel time. Travel time corresponds to the complete cycle of a robot within the storage area ; it includes the storage time from the picking station to the storage location, the interleaving time from the storage location to the next retrieval location, and the retrieval time from the retrieval location to the picking station. Rimélé et al. [67] present a model as a Markov Decision Process where a step corresponds to a state of the system when a robot is available at a picking station, and a decision has to be made between two types of tasks : a regular storage task or an opportunistic task. The latter corresponds to a situation where the shelf that is currently being carried by the robot can be used to fulfil another order instead of returning to the storage area. A regular storage task consists of selecting a storage location that is currently available. When a robot has stored a shelf, it retrieves the next assigned shelf and brings it to a picking station's waiting queue. Orders are stochastic and can enter the list of orders to fulfil at any moment, with any priority level (an Amazon *Prime* order would, for instance, be pushed to the top of the list). As such, the system needs to adapt to new orders continuously (no batching). While the future list of orders is uncertain at the decision step, the list of already-revealed orders is available. Several assumptions are made and listed below.

- A1. Only one picking station is considered.

- A2. The arrival of new orders is stochastic, but travelling and processing times are deterministic.

- A3. Opportunistic tasks, as described above, are automatically enforced when possible.

- A4. Since the focus is on the storage allocation problem, incoming orders are directly associated with a shelf. The sequencing is updated according to the order's deadline, with the condition that the corresponding shelf is not currently being carried by another robot.

- A5. Congestion and path optimisation for the robots are ignored.

- A6. Notions of stock levels and replenishment tasks are not considered.

- A7. Decisions have to be made swiftly, but we have access to a simulator that can be used offline to accumulate valuable experience. The simulator has the ability to copy its current state to run alternative future outcomes (with the essential condition of not revealing unknown information).

Following these assumptions about the real-time storage allocation problem, Section 6.4 presents the different methods developed in this work to tackle the problem.

6.4 Methodology

This section presents different methods, each with its advantages and drawbacks in terms of performance and computing time. All of these methods use the same base policy, which is learned offline from an expert system, namely an MCTS algorithm. Section 6.4.1 describes how we adapted an MCTS algorithm to the storage allocation problem. Using the experience accumulated from this MCTS algorithm, Section 6.4.2 explains how to learn a fast and efficient policy. Section 6.4.3 improves the learned policy's performance by using rollout strategies and a supervised MCTS. Finally, Section 6.4.4 presents a new algorithm that we call Supervised Tree Search (STS), which leverages the learned policy by exploring the search tree differently.

6.4.1 Monte Carlo Tree Search algorithm

Monte Carlo Tree Search is a method to optimise sequential decisions, relying on sampling to build an exploration tree, in which a node represents an action. It was initially designed for finite and binary decision games, but it is applicable – with adjustments – to any sequential decision problem [82]. Using random sampling to approximate decisions' values from Monte Carlo methods, Coulom [77] first proposed a Monte Carlo Tree Search algorithm. The idea is to run simulations until the end of the game and to use experience gained to incrementally build a tree, one node at a time. Within the tree, the nodes, or decisions, are selected based on the probability that they will perform best. The rest of each simulation makes decisions randomly, until a terminal state is reached. As depicted in Figure 6.1, a general MCTS iteration contains the following four steps : *Selection*, *Expansion*, *Simulation* and *Backpropagation*. Kocsis and Szepesvári [83] adapt a selection rule from the multi-armed bandit domain, namely the *Upper Confidence Bound (UCB1)* of Auer et al. [84], to MCTS and call it *Upper Confidence Bounds for trees (UCT)*. Inside the tree, UCT selects the child node i that maximizes $\bar{V_i} + 2c\sqrt{\frac{ln(N)}{n_i}}$, where $\bar{V_i}$ is the running average return value of node i, N is the number of visits of the parent node, n_i is the number of visits of node i, and c is

the exploration parameter (the higher the value of c, the greater the likelihood of selecting a less-visited node).

Figure 6.1 presents an iteration of the MCTS algorithm as used in this work. Unlike win-or-lose games, our system evolves in an infinite horizon and receives continuous rewards (the travel time of the robot) at every step. To adapt MCTS in our case, we first truncate the horizon after h steps. Second, to maintain the current average return value between 0 and 1, and to keep the two terms within the same order of magnitude, we propose to adapt the UCT equation with Eq. 6.1, where $\bar{V}$ represents the running average return value of the parent node.

$$\text{UCT} = \underset{i}{\text{argmax}} \frac{1}{\bar{V}_i} + \frac{c}{\bar{V}} \sqrt{\frac{ln\,(N)}{n_i}} \tag{6.1}$$

Note that $\bar{V}_i$ would only be equal to 0 if the decision of node i corresponds to an opportunistic task. Because we systematically enforce such tasks, UCT does not apply. For all other tasks, a cycle takes at least 1 second, so that $\frac{1}{\bar{V}_i}$ is always between 0 and 1 in Eq. 6.1.

In our system, when MCTS is executed, a sequence of decisions results in a deterministic output. This is explained by the fact that only revealed orders are considered in the trajectories, and travelling and processing times are assumed to be deterministic. This observation justifies two notable changes from a common MCTS. First, as depicted at the *Expansion* step in Figure 6.1, and as described in [82], a full node set expansion is performed. If k sibling nodes would normally be visited, this saves simulation time corresponding to the number of steps inside the tree. Indeed, this sub-trajectory would result in the same, redundant outcome before each child's first visit. Second, like in [85], at each node, we track not only the current average return value but also the best encountered simulation return value. The average value is still used for the selection step to explore more promising areas of the tree, but the actual final choice of action will select the node with the best encountered value. Empirically, these design choices gave the best results in our simulation study.

Finally, we use the Shortest Leg (SL) policy as the base policy at the *Simulation* step, instead of a Random policy. Indeed, as in [81], the SL storage policy is a relatively efficient and quickly deployable policy.

6.4.2 Learning a policy

While the MCTS algorithm presented above is capable of excellent performance to minimise the average cycle time, its performance is strongly dependent on the number of trajectories

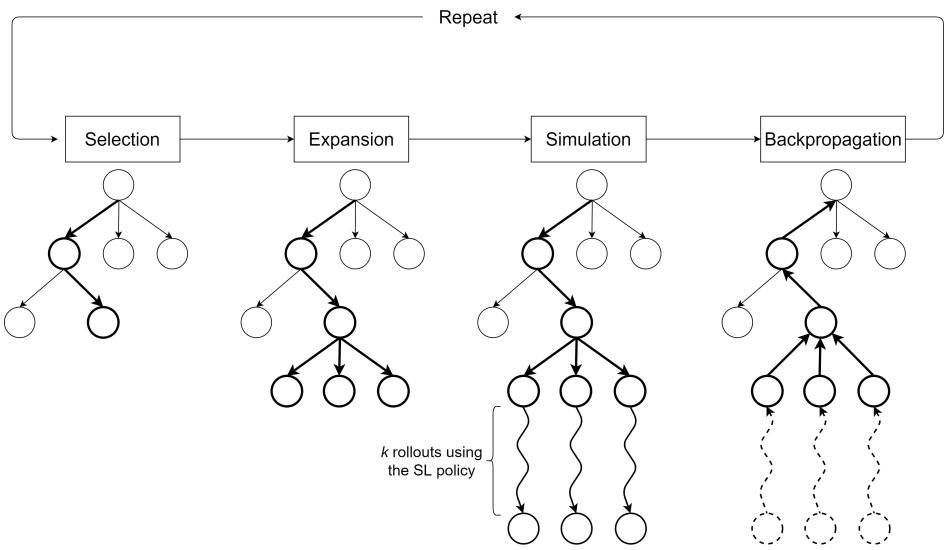

Figure 6.1 MCTS - The Selection step uses a selection rule (e.g., UCT) to select nodes (actions) within the tree until a leaf node is reached. The Expansion step creates one child or several children nodes from the current leaf node. The Simulation step runs simulations (trajectories) from a newly-created node using a fast base policy, either until the end of the game or for a given number of steps. Finally, the Backpropagation step updates the return value of the visited nodes based on the rewards revealed along the trajectories.

it runs at every decision step. The number of trajectories is directly correlated to the running time, limiting the opportunity for a satisfactory real-time deployment. Instead, we propose to use MCTS offline, when computing time is not an issue, to accumulate high-quality experience. Different versions of MCTS, corresponding to different truncated horizons h, are used to generate datasets of experiences that we refer to as *MCTS(h)*. The content of these datasets is described below.

Features representation

The complete state of the system at a decision point contains a tremendous number of variables [67], which would make any type of learning particularly difficult. Instead, we propose to define a Partially Observable Markov Decision Process (POMDP), where the observable state of the system corresponds to a vector of a limited number of key features. These features are chosen based on the expectation that they would provide sufficient information to make good decisions.

Figure 6.2 presents a simplified example of a warehouse containing six storage locations, four stored shelves and three waiting orders. Two locations are currently available, and the

sequence in which the revealed orders must be retrieved is known and written on the corresponding shelves.

The extracted features are two-fold. First, we represent which locations are available. If the warehouse is full (i.e., the same number of shelves and locations) and uses r robots (four in the example), the maximum number of available locations at any time step is r. These locations are ordered by increasing id and represented in the features vector by their (x, y) coordinates in the warehouse. If less than r locations are available, a mock location of coordinates $(-1, -1)$ is used to fill the remaining reserved features. Similarly, the next h orders are each represented by the corresponding shelves' coordinates. In the example, the resulting features vector would be :

$$[\underbrace{(1,1)\ ,\ (3,3)\ ,\ (-1,-1)\ ,\ (-1,-1)}_{\text{available locations}}\ ,\ \underbrace{(1,2)\ ,\ (3,2)\ ,\ (3,1)}_{\text{next } h \text{ orders' locations}}]$$

Using coordinates instead of location ids is preferable for generalisation during the learning. In general, if r denotes the maximum number of available locations (here equivalent to the number of robots) and h the number of revealed orders considered, the features vector contains $2(r + h)$ features. Finally, a features vector is saved in the dataset at every decision step, along with the id of the storage location decided by the MCTS method. Note that other features representations are worth considering, but this one gave the best results in terms of classification accuracy and computing time (see next paragraph and Section 6.5).

Learned Policy

A dataset MCTS(h) can now be used to learn a policy. This learning consists in having a classifier that takes as input a features vector, as previously described, and predicts the

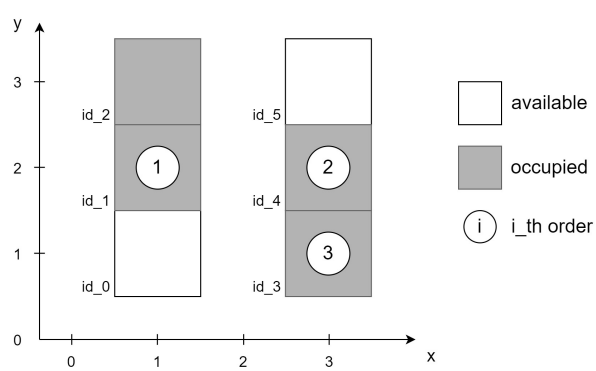

Figure 6.2 Simple features representation illustration

probabilities of each storage location to be selected. This classifier is denoted as $f : \mathbb{R}^n \mapsto [0,1]^{|\mathcal{L}|}$, where n is the number of features and $\mathcal{L}$ the set of storage locations. From this classifier, a *Learned Policy (LP)* can be defined as : $LP(X) = \underset{l}{\operatorname{argmax}} \left(f(X)_l \mid l \in \mathcal{L}^{\text{avail}} \right)$, where $X \in \mathbb{R}^n$ is a features vector, $f(X)_l$ refers to the lth element (location) of $f(X)$, and $\mathcal{L}^{\text{avail}}$ is the subset of the locations that are currently available. In other words, at every decision step, LP extracts a features vector from the current state of the system, and selects the available storage location with the highest predicted probability according to the classifier.

6.4.3 Enhancement of the Learned Policy

One advantage of a policy like LP is that it is extremely fast to deploy since it immediately selects an action based on the current state representation. However, by construction (supervised learning on a heuristic output) and due to the nature of the system (stochastic and partially observable), such a policy is necessarily sub-optimal. As described in [79], a lookahead strategy can be employed to take better advantage of a base policy. As used by Rimélé et al. [81] in a similar setting, a simple yet efficient approach is to perform a one-step lookahead with truncated rollouts. This approach runs simulations using already-revealed orders on every feasible action from the current state and deploys the base policy over $h-1$ more steps. When the simulation is completed, the initial action that resulted in the best outcome is selected. We refer to the resulting policy as *Base Policy+rollouts* ; for instance, LP+rollouts.

A natural extension of this approach is to consider multi-step lookaheads, for which MCTS can be seen as a special construction method. The Learned Policy can be used to guide the Selection step in MCTS. Silver at al. [86] in their famous AlphaGo program use a selection rule that combines a state value estimate and a policy network prediction. Here, we propose to only use the policy network prediction, but to sample the action based on the predicted values instead of taking the argmax. As such, if the parent node presents a features vector X, the selection rule is :

Supervised Selection(X) = Sample $\left(\{a \in A(X)\}, Prob(a) = \frac{pred(a)}{n_a+1} \right)$, where *Sample* is a sampling operator, $A(X)$ denotes the feasible actions from state X, and $pred(a)$ represents the filtered prediction from the Learned Policy : $pred(a) = \frac{f(X)_a}{\sum_{a' \in A(X)} f(X)_{a'}}$.

Although it performs well (see Section 6.5), the construction of a supervised MCTS may be limited by the lack of computing time for real-time deployment. First and fundamentally, MCTS iteratively builds a search tree running numerous trajectories, which takes time. As a result, it may not branch very deep without a significant number of trajectories. Second,

while a regular MCTS uses a fast heuristic rule to select nodes, our supervised MCTS uses a machine learning predictor. Each prediction is costly, mainly because the predictions need to be made sequentially, not in batches. Considering the valuable insights provided by the action classifier, we feel that another approach could be more beneficial. We propose such an approach below.

6.4.4 Supervised Tree Search algorithm

This algorithm's objective is to deeply explore the search tree with only a limited number of trajectories. To do so, we consider having an action classifier, here the Learned Policy, in which we can put significant trust.

The intuitive idea is the following. First, the Learned Policy is run until a maximum depth of h is reached. At each decision point, LP selects the feasible action of maximal prediction value. Along this trajectory, children nodes that were not selected are labelled as *open* nodes. For the following iterations, let us suppose that LP draws actions based on their prediction values instead of acting greedily. In that case, every open node has some probability of being reached by LP. This probability can be calculated as the node's prediction value times the probability of the parent node. Every iteration consists of exploring the search tree starting from the open node with the highest probability. From this node, the new trajectory will apply LP and open new nodes. The process is repeated until the maximum number of trajectories is reached. Figure 6.3 shows a small example of three iterations of the algorithm.

To allow some exploration, and not simply the exploitation of the action classifier, we propose to transform the prediction values by performing the following operation : $pred'_i = \frac{pred_i + (0.5 - pred_i) \times \eta}{\sum_{j \in \Gamma_k^+} pred'_j}$, $i \in \Gamma_k^+, \eta \in]0, 1[$, where Γ_k^+ designates the children nodes of node k and η is a parameter. This transformation tends to push all values toward equal probabilities, and because it is strictly increasing (when $\eta < 1$), the transformation does not change the predicted values' relative order. In this way, all actions keep a certain probability of being visited, even in the eventuality of the classifier predicting a zero probability. Results from the simulation study were rather invariant to small values of η. A value of $\eta = 0.1$ is used for the results presented below.

Algorithm 6.1 formalises the procedure. $\mathcal{N}$ designates the set of open nodes, and N_0 is the root node corresponding to the current state. r_N represents the immediate reward received when taking the action leading to node N, and V_N is the value of the current best trajectory starting at node N. Γ_N^+ denotes the set of children nodes (feasible actions) of node N, and the parent of node N is represented by Γ_N^-. A parameter depth_N is associated to each node

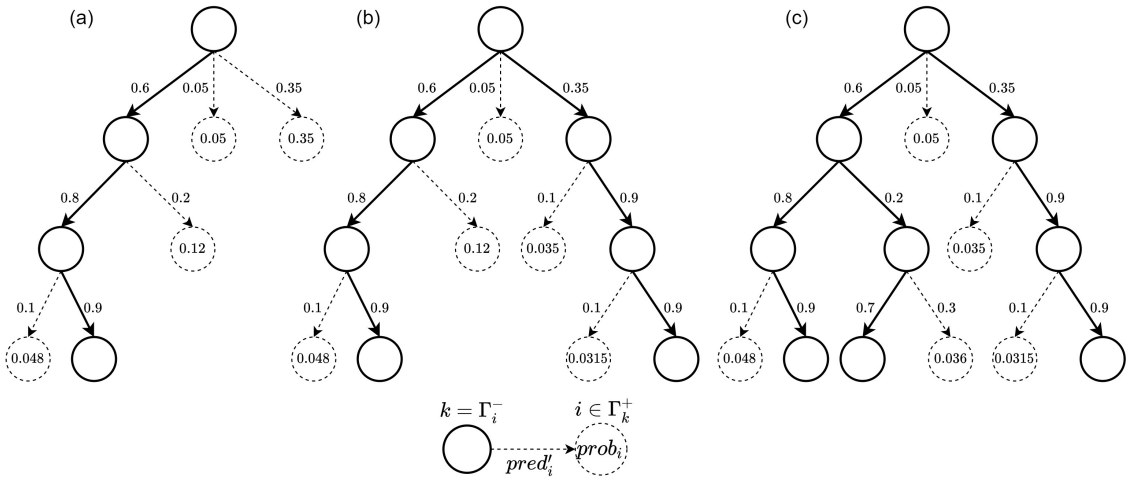

Figure 6.3 Supervised Tree Search - Step (a) corresponds to applying the Learned Policy starting from the initial state. Dashed lines represent the four nodes that were not selected along the trajectory. The value at the center of a node represents its visit probability, while the value beside an arrow represents the corresponding action's predicted value. Step (b) identifies the furthest right node as the open node with the highest visit probability. A new trajectory starts from this node and opens new nodes. In step (c), the node with the highest visit probability has a value of 0.12, and the resulting trajectory leaves one node open.

N to designate the depth of the node within the tree. Finally, γ denotes the discount factor applied to future return values.

The algorithm is decomposed in three functions : STS, LEARNED_POLICY and BACKPROP. STS corresponds to the main function and takes as argument the root node N_0 associated with the current state of the system. First, the set of open nodes $\mathcal{N}$ is initialized with the singleton N_0 (line 2), and the probability of this node to be reached by LP is trivially set to 1 (line 3). In line 4, a **while** loop is launched and run until the available computational budget is reached. At each iteration, the open node N_s with the highest probability of being reached by LP is selected (line 5). N_s is given as an argument to the LEARNED_POLICY function (line 6) before being removed from the set of open nodes (line 7). Finally, after the **while** loop has finished, STS returns the child node of the root node that resulted in the minimal estimated value (line 8).

The LEARNED_POLICY function is in charge of running LP until the maximum depth is reached in the tree. If the depth of node N_s received as an argument is lower than the maximum depth (line 10), the child node from N_s with the highest prediction value as determined by the action classifier is selected (line 11). The reward associated with this child node N_g is set to the reward received by the system at the step between N_s and N_g (line 12).

For all children nodes of N_s (line 13), their probabilities of being reached by LP are calculated (line 14), and their estimated values are set to $+\infty$. Children nodes different from N_g are added to the set of open nodes (lines 16-17). At line 18, the LEARNED_POLICY function is recursively run on the selected node N_g. If node N_s had reached the maximum depth (line 19), the rewards are backpropagated inside the tree with the BACKPROP function.

This BACKPROP function takes as arguments a node N and a return value R (line 21). If the parent node of N, Γ_N^-, currently has a greater (being worse in this context) estimated value $V_{\Gamma_N^-}$ than its associated reward $r_{\Gamma_N^-}$ plus the discounted return value R, then the estimated value is updated (lines 22-23). Finally, if the parent node Γ_N^- is not the root node N_0, the newly estimated value $V_{\Gamma_N^-}$ is backpropagated again from node Γ_N^-.

Algorithm 6.1 Supervised Tree Search (STS)

1: **function** STS(N_0)
2: $\quad \mathcal{N} = \{N_0\}$
3: $\quad \text{prob}_{N_0} = 1$
4: $\quad$ **while** within the computational budget **do**
5: $\quad\quad N_s = \underset{N}{\text{argmax}}\,(\text{prob}_N, N \in \mathcal{N})$
6: $\quad\quad$ LEARNED_POLICY(N_s)
7: $\quad\quad \mathcal{N} = \mathcal{N} \setminus N_s$
8: $\quad$ **return** $\underset{N}{\text{argmin}}\left(V_N, N \in \Gamma_{N_0}^+\right)$
9: **function** LEARNED_POLICY(N_s)
10: $\quad$ **if** $\text{depth}_{N_s} < h_{max}$ **then**
11: $\quad\quad N_g = \underset{N}{\text{argmax}}\left(\text{pred'}_N, N \in \Gamma_{N_s}^+\right)$
12: $\quad\quad r_{N_g} = \text{step}(N_s \rightarrow N_g)$
13: $\quad\quad$ **for all** $N \in \Gamma_{N_s}^+$ **do**
14: $\quad\quad\quad \text{prob}_N = \text{pred'}_N \times \text{prob}_{N_s}$
15: $\quad\quad\quad V_N = +\infty$
16: $\quad\quad\quad$ **if** $N \neq N_g$ **then**
17: $\quad\quad\quad\quad \mathcal{N} = \mathcal{N} \cup \{N\}$
18: $\quad\quad$ LEARNED_POLICY(N_g)
19: $\quad$ **else**
20: $\quad\quad$ BACKPROP(N_s, r_{N_s})
21: **function** BACKPROP(N, R)
22: $\quad$ **if** $V_{\Gamma_N^-} > r_{\Gamma_N^-} + \gamma\, R$ **then**
23: $\quad\quad V_{\Gamma_N^-} = r_{\Gamma_N^-} + \gamma\, R$
24: $\quad\quad$ **if** $\Gamma_{\Gamma_N^-}^- \neq \varnothing$ **then**
25: $\quad\quad\quad$ BACKPROP($\Gamma_N^-, V_{\Gamma_N^-}$)

6.5 Simulation study

This section presents the results of the various methods presented above. The required computing time differs widely from one method to another, and some parameters, such as the number of trajectories or the time horizon, greatly impact the results. This influence is extensively studied. Before presenting the policies' results, Section 6.5.1 presents preliminary results from the learning stage that are relevant in order to construct the final policies.

6.5.1 Preliminary results

Figure 6.4 represents the warehouse used in the simulation study. It contains 36 storage locations for as many shelves, and five robots are operating. The other parameters are similar to the ones described in [81]. A skewed demand distribution is considered, controlled by the cumulative distribution $G(x) = x^s \quad \text{for } 0 < s \leq 1$, where x represents $x\%$ of the shelves and s controls the skewness level. In this study, we set $s = 0.7$ for all the tests.

First, four MCTS(h) datasets were generated for h=5, 10, 20, and 30. By trial and error, the exploration parameter c in Section 6.4.1 was set to 1/16. Each dataset contains 3 million instances, and because there are five robots in the warehouse, the number of features per instance is equal to $2 \times (5 + h)$, so 20, 30, 50 and 70, respectively. Independent from the dataset, the same neural network architecture was used for the supervised learning task. The network is feed-forward and fully connected ; it contains five ReLU hidden layers of 100 neurons and a softmax output layer with 36 neurons, one for each storage location. The categorical cross-entropy was used as the loss function, and the Adam optimiser was used. The network was implemented using the Keras library [87]. Each training performed 500 episodes with mini-batches of 1024 instances.

For each dataset, 10% of the instances were reserved for a test set ; the resulting accuracies were 87.16%, 82.59%, 77.25% and 76.46%. Because of the stochasticity of the transition function and the heuristic nature of the MCTS method, a longer horizon leads to less predictable actions, making the learning task more difficult. However, the longer the horizon, the higher the potential to learn a good-performing policy. Indeed, when testing MCTS with 500 trajectories over different horizons on the test instances in Section 6.5.2, the corresponding average cycle times are 33.99s, 32.97s, 32.03s and 31.46s. The relationship between learning accuracy and performance creates a trade-off between learning a poor policy better and learning a better policy poorly. The following section will investigate this topic.

While learning from different policies results in different potential performances, one may wonder about the impact of the accuracy when learning from the same policy. Indeed, ac-

Figure 6.4 Plan view of the storage area

curacy is not necessarily proportional to the learned policy's performance, especially if the worst actions are well-learned and avoided. Figure 6.5 presents a study of the correlation between prediction accuracy and the performance of the learned policies. The y-axis on the left corresponds to the Learned Policy and the y-axis on the right corresponds to LP(5)+rollouts h=30 and STS(5) h=30 #traj=100. The learning was done on MCTS(5), and the results were generated from 10 instances of 4000 actions each. Smaller neural networks were used to degrade the classification accuracy. The relationship appears to be increasing monotonically, up to until an accuracy of 84%, which shows that accuracy is a good indicator to guide the network search. Above 84%, LP's performance seems to be inconsistent, as if a higher accuracy was over-compensated by misclassifications having a stronger impact in the objective function. However, the other learned policies' exploration gives a more consistent performance by reaching some plateau value. This result is of interest in this study when deciding on a network's architecture. Since the network needs to make sequential predictions used in a real-time policy, computing time is critical, and this time is typically dependent on the complexity of its architecture. If two networks present marginal accuracy differences, the simpler, smaller one will be favoured. In particular, we tested a convolutional neural network using a similar input but shaped so that each location corresponds to an element in a matrix. The obtained accuracy was 0.35% higher than the standard network, but the policies' computing time almost doubled.

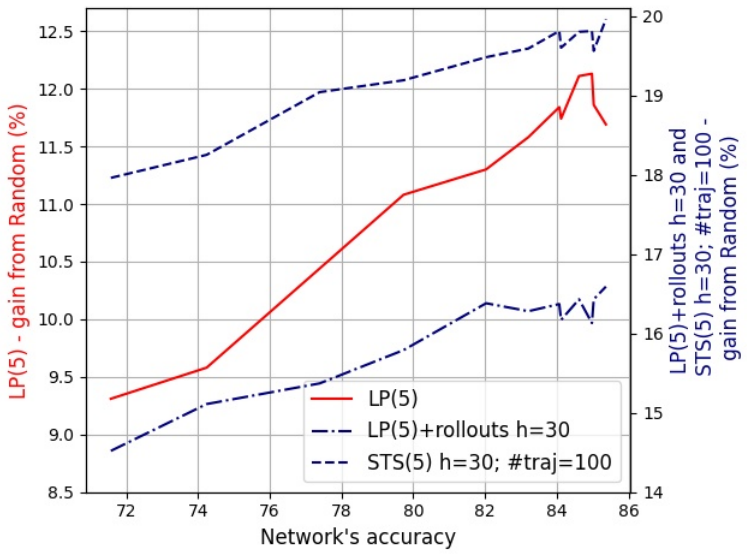

Figure 6.5 Correlation between the network's accuracy and the policies' performance

6.5.2 Results

The graph in Figure 6.6 presents the performance of the different storage policies by plotting the performance gain from the Random policy versus the average computing time per decision. All policies were tested over 100 instances of 4000 actions each. The different curves regroup families of policies, which only differ by a parameter value. Each mark along a curve corresponds to a different parameter mentioned in the legend. SL+rollouts and LP+rollouts have their maximum horizon vary up to 60. The curves representing offline MCTS, Supervised MCTS, and STS vary the number of trajectories used in these methods. The horizon of Supervised MCTS is set to 30. Because STS performs better, and for comparison purposes with offline MCTS, STS and offline MCTS have two curves each, corresponding to horizons of 30 and 60.

Only the methods learned on MCTS(10) are presented in this graph. This decision is justified by the results presented in Figure 6.7, where the policies learned from different datasets MCTS(h) are compared. The general assessment is that the policies improve when learning on longer-horizon datasets (for which the corresponding policies are also better). However, there is a noticeable exception with the Learned Policy that sees its performance decrease after $h = 10$. It seems that for $h > 10$, the classifiers make more mistakes (as expected with their accuracies), which degrades the policy. However, these classifiers still carry insightful information as they generate better than MCTS(10) results when more exploration is performed, particularly with STS. Overall, because of the good performance and simplicity

of MCTS(10), and the marginal gains of MCTS(20) and MCTS(30), we use MCTS(10) for comparisons with the other methods.

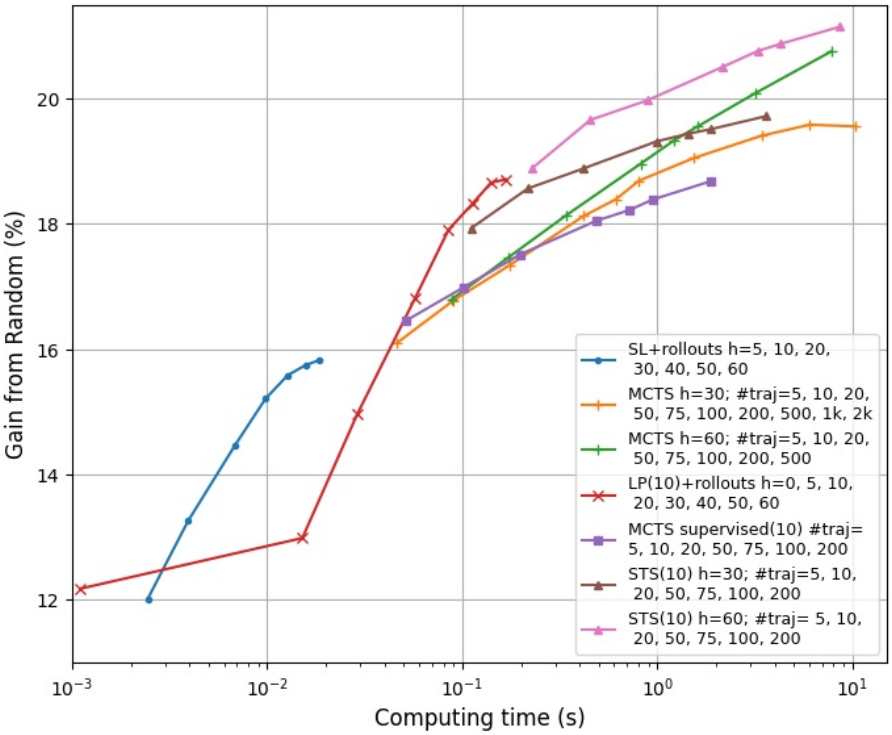

Figure 6.6 Policies' performance

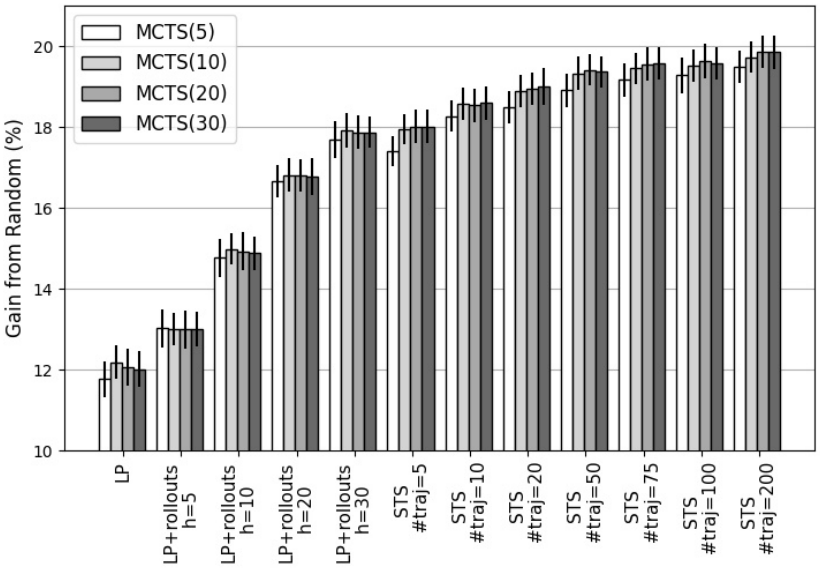

Figure 6.7 Learned policies' performance

For reference, and since it does not appear in Figure 6.6, SL improves the Random policy by 7.43% and takes 1.67e−4 sec. per action. Like SL, which does not perform any exploration, LP performs significantly better with a gain of about 12% in 1.10e−3 sec. Using rollouts improves both policies ; in particular, rollouts of horizon 30 generate gains of 15.2% and 17.9%, and rollouts of horizon 60 generate gains of 15.8% and 18.8%. These policies are particularly fast to deploy, with a computing time of less than 0.2 sec./action.

The family of tree search methods is computationally more expensive, and their computing time is heavily dependent on the number of trajectories run in the exploration. Supervised MCTS appears at first to perform slightly better at similar computing times, and then worse than offline MCTS when more trajectories are run. It seems that the classifier helps find reasonably good trajectories faster, but quickly reaches its limits when more trajectories are allowed.

STS is the best-performing method overall when more computing time is available. For a horizon of 30, and compared to MCTS with similar computing time, STS runs significantly fewer trajectories (about half) but consistently performs better. With a similar performance, STS runs much fewer trajectories than MCTS. For instance, 10 trajectories with STS results in a similar performance to MCTS with 75 to 100 trajectories, for a quarter of the computing time. The additional time comes from the predictions of the neural network. With only 5 trajectories, STS generates gains of 17.93% and MCTS 16.10%. With 200 trajectories, those gains become 19.72% and 19.05%, respectively. With a computing time of one second, STS gives gains of about 19.35% and MCTS 18.80%.

With a horizon of 60, 5 and 200 trajectories for STS give results of 18.89% and 21.15%, while for MCTS the results are 16.79% and 20.09%, respectively. With one second of computing time, STS and MCTS generate gains of about 20.10% and 19.10%, respectively.

The final choice of policy depends on the available time at the decision-making step. If the time is extremely limited (e.g., 0.001 sec.), LP would be the first choice. With more time (up to 0.1), LP+rollouts appears to be the best choice. If even more time is available, STS gives the best results. Detailed results of all the mentioned policies and their variants are given in the Appendix.

6.6 Conclusion

This work studied the real-time problem of allocating storage locations in a Robotic Mobile Fulfillment System. The proposed methods rely on learning a policy from the experience accumulated by a Monte Carlo Tree Search algorithm run offline. The neural network used

to learn the policy takes as input a features representation of the warehouse's current layout and a list of revealed orders. It predicts the probabilities of allocating a shelf to the storage locations. These probabilities can be used right away to deploy a speedy and well-performing Learned Policy (LP), or used in an exploration strategy. Three such strategies are presented, two of them generating convincing results in terms of performance and computing time. The first one performs lookahead rollouts on each feasible action, using LP as the rollout policy. This approach significantly improves the performance of LP while being very fast to deploy. The second strategy, referred to as Supervised Tree Search, builds a search tree by exploring in priority nodes with the highest probability of being reached by the Learned Policy. This method takes a longer computing time but further improves LP's performance and consistently beats MCTS. The policy choice ultimately depends on the time available for decision-making, and may differ from one warehousing operation to another.

There are several avenues for future research. First, some assumptions made in this work could be relaxed, such as considering several picking stations, notions of robots congestion, and others. A larger warehouse and robot fleet could also be considered to validate the proposed methods' scalability. Another research direction would be to expand on the supervised learning and tree search by integrating it with a Reinforcement Learning algorithm. This could further improve the policy network by leveraging the possibility of learning from higher-quality experiences obtained from the current version of STS.

Appendix : Detailed results

Table 6.1 Results of the methods without learning

4000 actions, avg. over 100 instances		h	# traj.	avg. cycle time (s)	st. dev. (s)	gain from Rand. (%)	# op. tasks	time (s)/action
No learning	Random	-	-	39.03	0.17	-	144.61	1.16e−4
	COL	-	-	38.25	0.19	1.99	143.11	1.14e−4
	Class-based	-	-	37.07	0.19	5.02	140.64	1.16e−4
	SL	-	-	36.14	0.20	7.42	143.90	1.16e−4
	SL + rollouts	5	-	34.34	0.16	12.02	151.76	2.44e−3
		10	-	33.86	0.17	13.26	182.32	3.93e−3
		20	-	33.39	0.15	14.46	181.87	6.82e−3
		30	-	33.09	0.16	15.21	181.19	9.80e−3
		40	-	32.95	0.16	15.59	180.34	1.27e−2
		50	-	32.89	0.16	15.75	181.39	1.57e−2
		60	-	32.85	0.17	15.83	180.85	1.86e−2
	MCTS	5	500	33.99	0.16	12.92	146.72	4.04e−1
		10	500	32.97	0.18	15.54	190.10	1.00
		20	500	32.02	0.14	17.95	201.33	2.06
		30	500	31.46	0.16	19.41	202.84	3.43
		30	5	32.75	0.16	16.10	185.26	4.61e−2
		30	10	32.48	0.16	16.78	189.25	9.00e−2
		30	20	32.26	0.14	17.34	192.14	1.75e−1
		30	50	31.96	0.16	18.13	195.66	4.17e−1
		30	75	31.85	0.15	18.39	198.45	6.12e−1
		30	100	31.74	0.16	18.69	198.65	8.04e−1
		30	200	31.60	0.15	19.05	201.58	1.54
		30	1000	31.39	0.13	19.58	203.66	6.04
		30	2000	31.40	0.15	19.56	203.59	10.38
		60	5	32.48	0.18	16.79	185.83	8.77e−2
		60	10	32.22	0.16	17.46	189.02	1.71e−1
		60	20	31.95	0.17	18.13	191.71	3.39e−1
		60	50	31.63	0.17	18.96	194.33	8.22e−1
		60	75	31.49	0.15	19.33	196.53	1.22
		60	100	31.40	0.16	19.56	197.98	1.61
		60	200	31.19	0.16	20.09	200.01	3.17
		60	500	30.93	0.17	20.75	203.46	7.79

Table 6.2 Results of the methods learned from MCTS(5)

4000 actions, avg. over 100 instances		h	# traj.	avg. cycle time (s)	st. dev. (s)	gain from Rand. (%)	# op. tasks	time (s)/action
MCTS(5)	LP	-	-	34.44	0.17	11.76	143.81	1.10e−3
	LP + rollouts	5	-	33.95	0.18	13.01	153.13	1.47e−2
		10	-	33.27	0.19	14.76	184.31	2.82e−2
		20	-	32.53	0.16	16.66	184.49	5.56e−2
		30	-	32.13	0.18	17.69	185.19	8.27e−2
		40	-	31.92	0.17	18.22	184.72	1.10e−1
		50	-	31.85	0.15	18.41	184.78	1.37e−1
		60	-	31.78	0.14	18.58	184.83	1.62e−1
	Supervised MCTS	30	5	32.64	0.17	16.38	181.26	5.10e−2
		30	10	32.48	0.16	16.79	180.95	9.93e−2
		30	20	32.34	0.13	17.15	181.26	1.92e−1
		30	50	32.14	0.16	17.66	183.14	4.63e−1
		30	75	32.03	0.17	17.93	184.03	6.73e−1
		30	100	31.96	0.17	18.12	184.75	8.83e−1
		30	200	31.81	0.16	18.50	186.28	1.71
	STS	30	5	32.24	0.15	17.41	174.27	1.06e−1
		30	10	31.90	0.15	18.26	183.81	2.07e−1
		30	20	31.82	0.16	18.49	186.22	4.01e−1
		30	50	31.66	0.16	18.90	189.97	9.24e−1
		30	75	31.56	0.16	19.16	192.88	1.34
		30	100	31.51	0.17	19.27	193.83	1.74
		30	200	31.43	0.16	19.49	196.76	3.32

Table 6.3 Results of the methods learned from MCTS(10)

4000 actions, avg. over 100 instances		h	# traj.	avg. cycle time (s)	st. dev. (s)	gain from Rand. (%)	# op. tasks	time (s)/action
MCTS(10)	LP	-	-	34.28	0.16	12.18	143.73	1.10e−3
	LP + rollouts	5	-	33.96	0.16	12.99	152.48	1.51e−2
		10	-	33.19	0.15	14.97	184.2	2.90e−2
		20	-	32.47	0.17	16.81	184.56	5.71e−2
		30	-	32.04	0.17	17.91	185.84	8.51e−2
		40	-	31.88	0.16	18.32	185.41	1.13e−1
		50	-	31.75	0.16	18.66	185.75	1.40e−1
		60	-	31.73	0.15	18.71	185.33	1.67e−1
	Supervised MCTS	30	5	32.61	0.18	16.46	183.13	5.17e−2
		30	10	32.40	0.16	16.99	183.2	1.01e−1
		30	20	32.20	0.17	17.52	184.99	1.99e−1
		30	50	31.99	0.16	18.05	186.91	4.83e−1
		30	75	31.92	0.16	18.22	187.3	7.14e−1
		30	100	31.85	0.19	18.39	189.48	9.49e−1
		30	200	31.74	0.16	18.68	190.75	1.87
	STS	30	5	32.03	0.14	17.93	178.14	1.11e−1
		30	10	31.79	0.15	18.57	184.86	2.17e−1
		30	20	31.66	0.16	18.89	189.24	4.19e−1
		30	50	31.49	0.16	19.31	192.66	9.87e−1
		30	75	31.44	0.15	19.44	193.61	1.44
		30	100	31.42	0.15	19.51	195.35	1.88
		30	200	31.33	0.15	19.72	197.44	3.61
		60	5	31.66	0.16	18.89	179.28	2.28e−1
		60	10	31.36	0.16	19.66	186.19	4.52e−1
		60	20	31.24	0.17	19.97	188.28	8.90e−1
		60	50	31.03	0.14	20.51	192.50	2.17
		60	75	30.93	0.15	20.76	193.55	3.26
		60	100	30.88	0.16	20.88	194.23	4.29
		60	200	30.78	0.14	21.15	197.11	8.64

Table 6.4 Results of the methods learned from MCTS(20)

4000 actions, avg. over 100 instances		h	# traj.	avg. cycle time (s)	st. dev. (s)	gain from Rand. (%)	# op. tasks	time (s)/action
MCTS(20)	LP	-	-	34.32	0.18	12.06	144.24	1.10e−3
	LP + rollouts	5	-	33.96	0.19	12.98	152.63	1.55e−2
		10	-	33.21	0.19	14.92	184.11	2.99e−2
		20	-	32.47	0.16	16.80	185.60	5.88e−2
		30	-	32.06	0.16	17.86	185.49	8.77e−2
		40	-	31.90	0.16	18.28	185.26	1.16e−1
		50	-	31.79	0.16	18.56	185.20	1.44e−1
		60	-	31.74	0.17	18.67	185.25	1.72e−1
	Supervised MCTS	30	5	32.59	0.17	16.51	183.99	5.28e−2
		30	10	32.40	0.15	16.98	185.70	1.03e−1
		30	20	32.20	0.16	17.49	185.96	2.01e−1
		30	50	31.99	0.18	18.05	188.58	4.87e−1
		30	75	31.89	0.15	18.31	189.36	7.22e−1
		30	100	31.84	0.15	18.43	190.65	9.55e−1
		30	200	31.73	0.17	18.72	191.76	1.88
	STS	30	5	32.01	0.16	18.00	179.06	1.16e−1
		30	10	31.80	0.16	18.54	185.96	2.26e−1
		30	20	31.64	0.15	18.94	188.62	4.39e−1
		30	50	31.46	0.15	19.41	193.67	1.04
		30	75	31.40	0.16	19.55	195.49	1.53
		30	100	31.37	0.17	19.62	195.58	2.26
		30	200	31.28	0.15	19.85	198.96	3.84

Table 6.5 Results of the methods learned from MCTS(30)

4000 actions, average on 100 instances		h	# traj	avg cycle time (s)	st dev (s)	gain from Rand. (%)	# op. tasks	time (s)/action
MCTS(30)	LP	-	-	34.35	0.18	12.01	144.59	1.15e−3
	LP + rollouts	5	-	33.96	0.17	12.99	152.44	1.61e−2
		10	-	33.23	0.16	14.88	184.24	3.08e−2
		20	-	32.49	0.18	16.77	184.15	6.07e−2
		30	-	32.06	0.15	17.86	185.41	9.03e−2
		40	-	31.88	0.18	18.31	184.94	1.20e−1
		50	-	31.77	0.17	18.61	184.65	1.50e−1
		60	-	31.74	0.16	18.69	183.84	1.79e−1
	Supervised MCTS	30	5	32.63	0.16	16.41	184.76	5.25e−2
		30	10	32.40	0.15	16.98	184.48	1.03e−1
		30	20	32.22	0.16	17.44	185.36	2.01e−1
		30	50	31.96	0.16	18.13	187.95	4.87e−1
		30	75	31.91	0.18	18.25	189.93	7.23e−1
		30	100	31.85	0.17	18.41	189.27	9.57e−1
		30	200	31.73	0.16	18.71	191.26	1.89
	STS	30	5	32.01	0.16	18.00	178.69	1.20e−1
		30	10	31.78	0.16	18.59	184.68	18.59
		30	20	31.62	0.18	19.00	188.97	4.56e−1
		30	50	31.48	0.15	19.36	192.71	1.08
		30	75	31.39	0.16	19.58	194.28	1.59
		30	100	31.39	0.15	19.58	195.17	2.07
		30	200	31.29	0.16	19.84	198.68	4.01

CHAPITRE 7 DISCUSSION GÉNÉRALE

Suivant la présentation des trois articles, un certain nombre d'enseignements peuvent être tirés des différents résultats obtenus.

Le premier article a permis de formaliser le problème de décision en temps réel et d'évaluer des règles de décision qui serviront de politiques de référence dans le reste de la thèse. Le deuxième et le troisième article en revanche emploient des méthodes d'apprentissage et d'exploration différentes, tout en gardant le même objectif de minimiser le temps moyen de déplacement des robots. Bien que les circonstances soient différentes (stockage par classes versus un stockage par emplacement exact, représentation de l'état courant, type d'exploration, etc.), certaines observations peuvent être faite, de manière comparative ou spécifiquement pour chaque approche.

L'approche d'apprentissage par renforcement, bien qu'ayant donné des résultats satisfaisants, n'a pas été simple à faire fonctionner. Tout d'abord, la façon dont l'apprentissage a été mené dans le deuxième article était très séquentielle, dans le sens où un simulateur unique était utilisé pour générer de l'expérience guidée par l'agent de renforcement. Cette expérience augmente graduellement en qualité mais cette amélioration est longue et nécessite un grand nombre d'épisodes d'apprentissage.

L'algorithme Q-learning est un agent de renforcement basé sur l'estimation de la *valeur* d'un état (*value-based*), à la différence d'autres agents qui optimisent directement les paramètres d'une politique (*policy gradient methods*). L'observation partielle donnée comme représentation de l'état du système a souffert d'un manque d'informations qui résulte en pratique en des transitions qui apparaissent stochastiques, alors qu'elles ne devraient pas l'être (exception faite de l'arrivée de nouvelles commandes). Par exemple, lorsqu'un robot "réserve" un emplacement pour une tâche de stockage, cet emplacement est immédiatement retiré des emplacements disponibles. En revanche, l'étagère qui sera par la suite récupérée par ce même robot occupe un emplacement qui apparaitra comme disponible seulement plusieurs étapes de temps plus tard (lorsque le robot aura véritablement libéré l'emplacement). Cette observation a semblé affecter plus fortement l'agent de Q-learning que l'approche supervisée du troisième article. Cela peut venir de la nécessité de l'agent de Q-learning d'apprendre la valeur d'un état, certainement plus variable que l'action idéale, ainsi que de l'évolution graduelle de la politique utilisée en ligne pour faire évoluer le système. Un autre exemple de transition faussement stochastique concerne exclusivement la méthode du troisième article. Dans cette méthode, une liste des k prochaines commandes déjà révélées est donnée dans

la représentation de l'état. Ces commandes sont ordonnées par ordre d'échéance croissant. Cependant, la liste exclue une commande (étagère) qui serait présentement transportée par un robot, puisque son emplacement est alors indéterminé. Cette commande apparaitra dans la liste dès que l'étagère correspondante aura été déposée ou qu'une tâche opportuniste se présentera.

Il est de l'opinion de l'auteur qu'une approche par apprentissage supervisé peut s'avérer être un bon choix de départ pour obtenir une politique de haute qualité dans un problème de décision tel que celui présenté dans cette thèse (si un besoin d'apprentissage se fait sentir en premier lieu). Une première condition est de disposer d'un simulateur qui permette une copie de son état courant afin d'être en mesure d'évaluer des trajectoires alternatives. La deuxième condition est de disposer d'un expert qui soit capable de déterminer une action de haute qualité si suffisamment de temps de calcul lui est alloué. Cet expert peut être une heuristique de recherche, telle que la recherche arborescente Monte-Carlo utilisée dans cette thèse, ou bien même une méthode exacte si la configuration du système le permet. L'expert peut alors être déployé en parallèle pour rapidement générer une quantité importante d'expériences de haute qualité. Ces expériences peuvent ensuite être apprises par un prédicteur. Cette opinion mériterait certainement une étude spécifique afin de définir une liste de critères associés aux problèmes de prise de décision qui ferait d'une approche par renforcement un premier choix.

CHAPITRE 8 CONCLUSION ET RECOMMANDATIONS

8.1 Synthèse des travaux

Cette thèse s'intéresse à un nouveau type d'entrepôt automatique spécialisé dans le commerce en ligne, appelé centre de distribution robotisé. Plus précisément, le sujet de recherche porte sur l'optimisation en temps réel des problèmes opérationnels dans un tel entrepôt, avec un intérêt particulier pour le problème d'allocation d'emplacements de stockage. La présentation de la thèse s'organise autour de trois articles dont les conclusions sont présentées ci-après.

Le premier article présente un modèle de programmation dynamique stochastique qui formalise la prise de décision opérationnelle en temps réel dans un centre de distribution robotisé. Plusieurs problèmes de décision sont considérés, incluant l'ordonnancement des commandes, l'allocation d'emplacements de stockage, la sélection d'étagères et l'affectation des commandes aux stations de cueillette. Évoluant dans un environnement concurrentiel extrêmement compétitif, les vendeurs en ligne tendent à promettre des livraisons toujours plus rapides tout en réduisant les coûts au maximum. Cela force la chaîne logistique à considérer les nouvelles commandes dès qu'elles sont révélées. Cet impératif de réactivité couplé à la dynamique du système de stockage rend la prise de décision particulièrement complexe. Ces problèmes sont conventionnellement traités par des règles de décision de haut niveau. La formalisation mathématique proposée dans cet article est destinée à assister le chercheur dans le développement d'approches de résolution plus avancées, ou du moins, à sensibiliser la recherche à la prise de décision en temps réel. Le modèle est illustré sur le sous-problème d'allocation d'emplacements de stockage en modélisant les règles de décision. Les résultats obtenus servent de points de référence pour les nouvelles politiques de stockage développées dans les autres articles.

Le deuxième article s'intéresse spécifiquement au problème d'allocation d'emplacements de stockage. Lorsqu'un robot est libéré à une station de cueillette, l'objectif est de sélectionner une zone de l'entrepôt où entreposer l'étagère afin de minimiser le temps de cycle moyen des robots. L'approche de résolution utilise un algorithme d'apprentissage par renforcement Deep Q-learning. Dans cet apprentissage, le système est présenté comme un processus de décision markovien partiellement observable où un état est représenté par de l'information sur le taux de demande de l'étagère à entreposer, l'emplacement de la prochaine commande et l'état d'occupation des zones. La politique obtenue améliore les meilleures politiques de référence de 3.21 à 4.83% en fonction de la distribution de la demande. Dans une seconde phase, afin d'utiliser plus d'information sur les commandes déjà révélées à un instant donné, une stratégie

d'exploration de trajectoires est proposée. Cette exploration est applicable à n'importe quelle politique, incluant les politiques de référence, et génère des gains de performance significatifs. La politique apprise voit ainsi sa performance augmentée de 2.78 à 4.54%.

Enfin, le troisième article étend l'exploration de trajectoires en s'intéressant à l'exploration d'arbres. L'objectif de cet article est toujours d'apprendre une politique de stockage qui minimise le temps de déplacement moyen des robots, à la différence que les emplacements sont ici sélectionnés exactement plutôt que de sélectionner une zone. Une recherche arborescente Monte-Carlo est utilisée hors ligne pour accumuler une importante quantité d'expériences de haute qualité. Étant donné le temps de calcul important d'une telle recherche arborescente, elle ne peut être déployée en temps réel. L'expérience accumulée est représentée avec les coordonnées des emplacements disponibles ainsi que celles des prochaines commandes révélées. Cette représentation est apprise par un réseau de neurones qui est en mesure de prédire l'emplacement recommandé pour chaque étagère. Le réseau de neurones peut être utilisé tel quel pour définir une politique apprise extrêmement rapide à déployer et performante (+4.7% par rapport à la meilleure politique de référence). Le réseau peut également être utilisé pour guider une exploration de trajectoires ou une recherche d'arbre. Une nouvelle recherche d'arbre supervisée est proposée. Cette recherche explore en priorité les régions de l'arbre de décision ayant la plus grande probabilité d'être atteintes par la politique apprise. Dépendamment du temps alloué à la prise de décision en temps réel, la recherche d'arbre supervisée surperforme la politique de référence de 12.4 à 14.8%.

8.2 Limitations et améliorations futures

En plus de certaines limitations des approches proposées dans ces travaux de recherche, plusieurs hypothèses ont été faites, dont certaines mériteraient d'être levées.

Dans le premier article, la modélisation suppose des commandes uniques qui ne contiennent qu'un seul type d'objet. Dans certains centres de distribution robotisés, la consolidation des commandes se fait dans un espace de l'entrepôt distinct des stations de cueillette. Cette hypothèse est alors tout à fait valide. En revanche, dans d'autres centres de distribution, la consolidation se fait directement à la station de cueillette. Il serait alors utile de proposer une modélisation qui considère explicitement les commandes multiples afin d'éviter des délais et des saturations au niveau des stations.

Les deuxième et troisième articles font des hypothèses semblables quant à la configuration de l'entrepôt et à la gestion des autres problèmes de décision. Pour ce qui est de la configuration de l'entrepôt, il serait intéressant de considérer plus qu'une station de cueillette puisque cela

changerait la notion d'accessibilité d'une étagère. Il serait également intéressant d'évaluer l'extensibilité des méthodes proposées à des espaces de stockage de plus grandes dimensions. La méthode du deuxième article discrétise l'espace en zones, ce qui devrait raisonnablement permettre une bonne extensibilité. Le troisième article en revanche sélectionne des emplacements exacts. Une discrétisation ou une restriction des emplacements considérés pourraient s'avérer nécessaires.

Pour ce qui a trait aux problèmes de décision autres que l'allocation d'emplacements de stockage, l'optimisation simultanée de l'ordonnancement des commandes et la sélection des étagères pourrait présenter une grande valeur ajoutée. Dans les méthodes présentées, l'ordonnancement des commandes se fait par ordre d'échéance croissant. Permettre un réordonnancement permettrait de créer des combinaisons de commandes pour des cycles plus efficaces, voire un plus grand nombre de cycles opportunistes. De même, dans ces travaux, une commande est automatiquement associée à une étagère. En réalité, un type d'article peut être trouvé dans plusieurs étagères. Être en mesure de sélectionner quelle étagère utiliser pour chaque commande pourrait également permettre la création de meilleurs cycles.

Un sujet qui n'a pas été abordé dans cette thèse, mais qui fait l'objets de plusieurs articles de recherche, concerne le problème de routage des robots afin d'éviter des congestions. Incorporer cette thématique avec les problèmes opérationnels serait intéressant d'un point de vue pratique.

Enfin, plusieurs extensions pourraient être faites par rapport aux méthodes de résolution. La représentation d'un état du système est essentielle à la réussite des techniques d'apprentissage. Bien que la représentation par coordonnées du troisième article ait donné de bons résultats, d'autres représentations pourraient être imaginées. Le type de modèle d'apprentissage est directement relié au choix de la représentation. Lorsqu'il est envisagé de réaliser une recherche de trajectoires en déploiement de la politique, il est important de considérer le coût computationnel du modèle d'apprentissage. Finalement, le réseau de politique obtenu dans le troisième article, qui utilisé dans une exploration d'arbre, pourrait être amélioré en utilisant la nouvelle politique comme génératrice de nouvelles expériences sur lesquelles entraîner le réseau (ce qui reviendrait à une approche de renforcement avec une exploration d'arbre).

RÉFÉRENCES

[1] F. Ali, “US ecommerce grows 44.0% in 2020 | Digital Commerce 360,” 2021. [En ligne]. Disponible : https://www.digitalcommerce360.com/article/us-ecommerce-sales/

[2] N. Boysen, K. Stephan et F. Weidinger, “Manual order consolidation with put walls : the batched order bin sequencing problem,” *EURO Journal on Transportation and Logistics*, vol. 8, n°. 2, p. 169–193, 2019.

[3] H. Yaman, O. E. Karasan et B. Y. Kara, “Release time scheduling and hub location for next-day delivery,” *Operations Research*, vol. 60, n°. 4, p. 906–917, 2012.

[4] N. Boysen, R. De Koster et F. Weidinger, “Warehousing in the e-commerce era : A survey,” *European Journal of Operational Research*, vol. 277, n°. 2, p. 396–411, 2019.

[5] R. Ramanathan, “The moderating roles of risk and efficiency on the relationship between logistics performance and customer loyalty in e-commerce,” *Transportation Research Part E : Logistics and Transportation Review*, vol. 46, n°. 6, p. 950–962, 2010.

[6] P. R. Wurman, R. D’Andrea et M. Mountz, “Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses,” *AI Magazine*, vol. 29, n°. 1, p. 9–9, 3 2008.

[7] K. Azadeh, R. de Koster et D. Roy, “Robotized Warehouse Systems : Developments and Research Opportunities,” *SSRN Electronic Journal*, p. 1–55, 2017.

[8] S. Banker, “Robots In The Warehouse : It’s Not Just Amazon,” *Forbes Business*, 2016. [En ligne]. Disponible : https://www.forbes.com/sites/stevebanker/2016/01/11/robots-in-the-warehouse-its-not-just-amazon

[9] C. A. Valle et J. E. Beasley, “Order allocation, rack allocation and rack sequencing for pickers in a mobile rack environment,” *Computers and Operations Research*, vol. 125, n°. 2021, p. 105090, 2021.

[10] R. D’Andrea et P. Wurman, “Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities,” *2008 IEEE International Conference on Technologies for Practical Robot Applications, TePRA*, p. 80–83, 2008.

[11] J. J. Enright et P. R. Wurman, “Optimization and coordinated autonomy in mobile fulfillment systems,” *AAAI Workshop - Technical Report*, vol. WS-11-09, p. 33–38, 2011.

[12] S. Banker, “New Solution Changes the Rules of Warehouse Automation,” *Forbes Business*, 2018. [En ligne]. Disponible : https://www.forbes.com/sites/stevebanker/2018/06/12/new-solution-changes-the-rules-of-warehouse-automation

[13] M. Merschformann, T. Lamballais, R. De Koster et L. Suhl, “Decision rules for robotic mobile ful fi llment systems,” *Operations Research Perspectives*, vol. 6, n°. March, p. 100128, 2019.

[14] K. J. Roodbergen et I. F. A. Vis, “A survey of literature on automated storage and retrieval systems,” *European Journal of Operational Research*, vol. 194, n°. 2, p. 343–362, 2009.

[15] H. Zollinger, “AS/RS application, benefits and justification in comparison to other storage methods : A white paper,” *Automated Storage Retrieval Systems Production Section of the Material Handling Industry of America*, p. 1–24, 2014.

[16] N. Boysen et K. Stephan, “A survey on single crane scheduling in automated storage/retrieval systems,” *European Journal of Operational Research*, vol. 254, n°. 3, p. 691–704, 2016.

[17] M.-H. Han, L. F. McGinnis, J. S. Shieh et J. A. White, “On Sequencing Retrievals In An Automated Storage/Retrieval System,” *IIE Transactions*, vol. 19, n°. 1, p. 56–66, 3 1987. [En ligne]. Disponible : http://www.tandfonline.com/doi/abs/10.1080/07408178708975370

[18] R. de Koster, T. Le-Duc et K. J. Roodbergen, “Design and control of warehouse order picking : A literature review,” *European Journal of Operational Research*, vol. 182, n°. 2, p. 481–501, 2007.

[19] J. L. Heskett, “Cube-per-order index-a key to warehouse stock location,” *Transportation and distribution Management*, vol. 3, n°. 1, p. 27–31, 1963.

[20] W. Hausman, L. Schwarz et S. Graves, “Optimal Storage Assignment in Automatic Warehousing Systems,” *Management Science*, vol. 22, n°. 6, p. 629–638, 1976.

[21] S. C. Graves, W. H. Hausman et L. B. Schwarz, “Storage-Retrieval Interleaving in Automatic Warehousing Systems,” *Management Science*, vol. 23, n°. 9, p. 935–945, 5 1977.

[22] Y. A. Bozer et J. A. White, “Travel-Time Models for Automated Storage/Retrieval Systems,” *IIE Transactions*, vol. 16, n°. 4, p. 329–338, 12 1984.

[23] L. B. Schwarz, S. C. Graves et W. H. Hausman, “Scheduling Policies for Automatic Warehousing Systems : Simulation Results,” *A I I E Transactions*, vol. 10, n°. 3, p. 260–270, 9 1978.

[24] R. J. Linn et R. Wysk, “A simulation model for evaluating control algorithms of an automated storage/retrieval system,” p. 330–339, 1984. [En ligne]. Disponible : https://dl.acm.org/citation.cfm?id=809482

[25] R. Linn et R. Wysk, “An Analysis Of Control Strategies For An Automated Storage/Retrieval System,” *INFOR : Information Systems and Operational Research*, vol. 25, n°. 1, p. 66–83, 1 1987. [En ligne]. Disponible : http://www.tandfonline.com/doi/full/10.1080/03155986.1987.11732029

[26] J. P. Van den Berg, “Class-based storage allocation in a single- command warehouse with space requirement constraints,” *International Journal of Industrial Engineering*, vol. 3, p. 21–28, 1996.

[27] J. P. van den Berg et A. Gademann, “Simulation study of an automated storage/retrieval system,” *International Journal of Production Research*, vol. 38, n°. 6, p. 1339–1356, 4 2000.

[28] J.-P. Gagliardi, J. Renaud et A. Ruiz, “A simulation modeling framework for multiple-aisle automated storage and retrieval systems,” *Journal of Intelligent Manufacturing*, vol. 25, n°. 1, p. 193–207, 2 2014.

[29] J.-P. Gagliardi, J. Renaud et A. Ruiz, “On storage assignment policies for unit-load automated storage and retrieval systems,” *International Journal of Production Research*, vol. 50, n°. 3, p. 879–892, 2 2012.

[30] F. H. Lee et S. K. Schaefer, “Sequencing Methods For Automated Storage And Retrieval Systems With Dedicated Storage,” *Computers & Industrial Engineering*, vol. 32, n°. 2, p. 351–362, 1997.

[31] S. Mahajan, B. V. Rao et B. A. Peters, “A retrieval sequencing heuristic for miniload end-of-aisle automated storage/retrieval systems,” *International Journal of Production Research*, vol. 36, n°. 6, p. 1715–1731, 1998.

[32] J. P. Van den Berg et A. J. Gademann, “Optimal routing in an automated storage/retrieval system with dedicated storage,” *IIE Transactions (Institute of Industrial Engineers)*, vol. 31, n°. 5, p. 407–415, 1999.

[33] A. Alonso-Ayuso, G. Tirado et A. Udias, “On a selection and scheduling problem in automatic storage and retrieval warehouses,” *International Journal of Production Research*, vol. 51, n°. 17, p. 5337–5353, 2013.

[34] Y. L. Yin et H. Rau, “Dynamic selection of sequencing rules for a class-based unit-load automated storage and retrieval system,” *International Journal of Advanced Manufacturing Technology*, vol. 29, n°. 11-12, p. 1259–1266, 2006.

[35] K. Hachemi, Z. Sari et N. Ghouali, “A step-by-step dual cycle sequencing method for unit-load automated storage and retrieval systems,” *Computers and Industrial Engineering*, vol. 63, n°. 4, p. 980–984, 2012. [En ligne]. Disponible : http://dx.doi.org/10.1016/j.cie.2012.06.009

[36] J.-P. Gagliardi, J. Renaud et A. Ruiz, “On sequencing policies for unit-load automated storage and retrieval systems,” *International Journal of Production Research*, vol. 52, n°. 4, p. 1090–1099, 2 2014.

[37] T. Wauters, F. Villa, J. Christiaens, R. Alvarez-Valdes et G. Vanden Berghe, “A decomposition approach to dual shuttle automated storage and retrieval systems,” *Computers and Industrial Engineering*, vol. 101, p. 325–337, 2016. [En ligne]. Disponible : http://dx.doi.org/10.1016/j.cie.2016.09.013

[38] A. Roozbeh Nia, H. Haleh et A. Saghaei, “Dual command cycle dynamic sequencing method to consider GHG efficiency in unit-load multiple-rack automated storage and retrieval systems,” *Computers & Industrial Engineering*, vol. 111, p. 89–108, 9 2017. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0360835217302978

[39] H. F. Lee et S. K. Schaefer, “Retrieval sequencing for unit-load automated storage and retrieval systems with multiple openings,” *International Journal of Production Research*, vol. 34, n°. 10, p. 2943–2962, 1996.

[40] M. Goetschalckx et H. D. Ratliff, “Shared Storage Policies Based on the Duration Stay of Unit Loads,” *Management Science*, vol. 36, n°. 9, p. 1120–1132, 9 1990. [En ligne]. Disponible : http://pubsonline.informs.org/doi/abs/10.1287/mnsc.36.9.1120

[41] P. Montulet, A. Langevin et D. Riopel, “Le Problème De L’Optimisation De L’Entreposage Partage : Méthodes Exacte Et Heuristique,” *INFOR : Information Systems and Operational Research*, vol. 35, n°. 2, p. 138–153, 1997. [En ligne]. Disponible : http://www.tandfonline.com/doi/full/10.1080/03155986.1997.11732323

[42] L. Chen, A. Langevin et D. Riopel, “The storage location assignment and interleaving problem in an automated storage/retrieval system with shared storage,” *International Journal of Production Research*, vol. 48, n°. 4, p. 991–1011, 2010.

[43] P. Yang, L. Miao, Z. Xue et B. Ye, “Variable neighborhood search heuristic for storage location assignment and storage/retrieval scheduling under shared storage in multi-shuttle automated storage/retrieval systems,” *Transportation Research Part E : Logistics and Transportation Review*, vol. 79, p. 164–177, 2015. [En ligne]. Disponible : http://www.scopus.com/inward/record.url?eid=2-s2.0-84929321553&partnerID=40&md5=368be9e67deda5b357636d9aacd59204

[44] T. Lamballais, D. Roy et R. De Koster, “Estimating performance in a Robotic Mobile Fulfillment System,” *European Journal of Operational Research*, vol. 256, n°. 3, p. 976–990, 2017.

[45] B. Zou, Y. Y. Gong, X. Xu et Z. Yuan, “Assignment rules in robotic mobile fulfilment systems for online retailers,” *International Journal of Production Research*, vol. 55, n°. 20, p. 6175–6192, 2017.

[46] R. Yuan, S. Graves et T. Cezik, “Velocity-Based Storage Assignment in Semi-Automated Storage Systems,” *Production and Operations Management*, vol. 28, n°. 2, p. 354–373, 2018.

[47] D. Roy, S. Nigam, R. de Koster, I. Adan et J. Resing, “Robot-storage zone assignment strategies in mobile fulfillment systems,” *Transportation Research Part E : Logistics and Transportation Review*, vol. 122, n°. April 2018, p. 119–142, 2019.

[48] T. Lamballais, D. Roy et R. De Koster, “Inventory allocation in robotic mobile fulfillment systems,” *IISE Transactions*, vol. 52, n°. 1, p. 1–17, 2020.

[49] N. Boysen, D. Briskorn et S. Emde, “Parts-to-picker based order processing in a rack-moving mobile robots environment,” *European Journal of Operational Research*, vol. 262, n°. 2, p. 550–562, 2017.

[50] A. Gharehgozli et N. Zaerpour, “Robot scheduling for pod retrieval in a robotic mobile fulfillment system,” *Transportation Research Part E : Logistics and Transportation Review*, vol. 142, n°. February, 2020.

[51] A. Bolu et O. Korcak, “Adaptive Task Planning for Multi-Robot Smart Warehouse,” *IEEE Access*, vol. X, p. 1–1, 2021.

[52] R. Krenzler, L. Xie et H. Li, “Deterministic Pod Repositioning Problem in Robotic Mobile Fulfillment Systems,” *arXiv preprint*, vol. 1810.05514, n°. April 2020, 2018.

[53] F. Weidinger, N. Boysen et D. Briskorn, “Storage assignment with rack-moving mobile robots in KIVA warehouses,” *Transportation Science*, vol. 52, n°. 6, p. 1479–1495, 2018.

[54] M. Mirzaei, N. Zaerpour et R. de Koster, “The impact of integrated cluster-based storage allocation on parts-to-picker warehouse performance,” *Transportation Research Part E : Logistics and Transportation Review*, vol. 146, n°. October 2020, 2021.

[55] X. Li, G. Hua, A. Huang, J. B. Sheu, T. C. Cheng et F. Huang, “Storage assignment policy with awareness of energy consumption in the Kiva mobile fulfilment system,” *Transportation Research Part E : Logistics and Transportation Review*, vol. 144, n°. November 2019, p. 102158, 2020. [En ligne]. Disponible : https://doi.org/10.1016/j.tre.2020.102158

[56] J. Zhang, F. Yang et X. Weng, “A building-block-based genetic algorithm for solving the robots allocation problem in a robotic mobile fulfilment system,” *Mathematical Problems in Engineering*, vol. 2019, 2019.

[57] L. Xie, N. Thieme, R. Krenzler et H. Li, “Introducing split orders and optimizing operational policies in robotic mobile fulfillment systems,” *European Journal of Operational Research*, vol. 288, n°. 1, p. 80–97, 2021.

[58] M. Merschformann, L. Xie et H. Li, “RAWSim-O : A simulation framework for robotic mobile fulfillment systems,” *Logistics Research*, vol. 11, n°. 1, p. 1–11, 2018.

[59] J. A. Tompkins et J. D. Smith, *The Warehouse Management Handbook, Second Edition.* Tompkins Press, 1998.

[60] J. Gu, M. Goetschalckx et L. F. McGinnis, “Research on warehouse operation : A comprehensive review,” *European Journal of Operational Research*, vol. 177, n°. 1, p. 1–21, 2007.

[61] B. Shah et V. Khanzode, “A comprehensive review of warehouse operational issues,” *International Journal of Logistics Systems and Management*, vol. 26, n°. 3, p. 346–378, 2017.

[62] H. Davarzani et A. Norrman, “Toward a relevant agenda for warehousing research : literature review and practitioners’ input,” *Logistics Research*, vol. 8, n°. 1, 2015.

[63] W. B. Powell, “A unified framework for stochastic optimization,” *European Journal of Operational Research*, vol. 275, n°. 3, p. 795–821, 2019.

[64] T. Kirks, J. Stenzel, A. Kamagaew et M. ten Hompel, “Cellular Transport Vehicles for Flexible and Changeable Facility Logistics Systems,” *Logistics Journal*, vol. 2192(9084), n°. 1, 2012.

[65] Y. A. Bozer et F. J. Aldarondo, “A simulation-based comparison of two goods-to-person order picking systems in an online retail setting,” *International Journal of Production Research*, vol. 56, n°. 11, p. 3838–3858, 2018.

[66] M. Guan et Z. Li, “Genetic Algorithm for Scattered Storage Assignment in Kiva Mobile Fulfillment System,” *American Journal of Operations Research*, vol. 08, n°. 06, p. 474–485, 2018.

[67] A. Rimélé, M. Gamache, M. Gendreau, P. Grangier et L.-M. Rousseau, “Robotic mobile fulfillment systems : a mathematical modelling framework for e-commerce applications,” *International Journal of Production Research*, p. 1–17, may 2021.

[68] C. Watkins, “Learning from delayed rewards,” Thèse de doctorat, King’s College, 1989.

[69] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar et D. Silver, “Rainbow : Combining improvements in deep reinforcement learning,” dans *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018, p. 3215–3222.

[70] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg et D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, n°. 7540, p. 529–533, 2015.

[71] B. C. Park, "Order Picking : Issues, Systems and Models," dans *Warehousing in the Global Supply Chain*. London : Springer London, 2012, p. 1–30.

[72] N. Boysen, S. Fedtke et F. Weidinger, "Optimizing automated sorting in warehouses : The minimum order spread sequencing problem," *European Journal of Operational Research*, vol. 270, n°. 1, p. 386–400, 2018.

[73] R. S. Sutton et A. G. Barto, *Reinforcement Learning : An Introduction*. MIT press, 2018.

[74] D. Bertsekas, "Dynamic Programming and Stochastic Control," *Mathematics in Science and Engineering*, vol. 125, p. 222–293, 1976.

[75] H. van Hasselt, "Double Q-learning," *Advances in Neural Information Processing Systems 23 : 24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010*, p. 1–9, 2010.

[76] H. van Hasselt, A. Guez et D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, p. 2094–2100, 2016.

[77] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," dans *Computers and Games*, H. J. van den Herik, P. Ciancarini et H. H. L. M. J. Donkers, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 72–83.

[78] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel et D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, n°. 7676, p. 354–359, oct 2017.

[79] D. Bertsekas, *Reinforcement learning and optimal control*. Athena scientific, 2019.

[80] J.-P. Gagliardi, J. Renaud et A. Ruiz, "Models for automated storage and retrieval systems : a literature review," *International Journal of Production Research*, vol. 50, n°. 24, p. 7110–7125, 12 2012.

[81] A. Rimélé, P. Grangier, M. Gamache, M. Gendreau et L.-M. Rousseau, "E-commerce warehousing : learning a storage policy," *arXiv preprint*, vol. 2101.08828, 2021.

[82] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis et S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, n°. 1, p. 1–43, 2012.

[83] L. Kocsis et C. Szepesvári, "Bandit based Monte-Carlo planning," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4212 LNAI, p. 282–293, 2006.

[84] P. Auer, N. Cesa-Bianchi et P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, n°. 2-3, p. 235–256, 5 2002.

[85] Y. Bjornsson et H. Finnsson, "CadiaPlayer : A simulation-based general game player," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, n°. 1, p. 4–15, 3 2009.

[86] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel et D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, n°. 7587, p. 484–9, 2016.

[87] F. Chollet *et al.*, "Keras," https://keras.io, 2015.