

Titre: Apprentissage Profond en Optimisation Combinatoire :
Title: Apprentissage de Bornes et Résolution de Problèmes de Transport

Auteur: Augustin Parjadis de Larivière
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Parjadis de Larivière, A. (2023). Apprentissage Profond en Optimisation Combinatoire : Apprentissage de Bornes et Résolution de Problèmes de Transport [Thèse de doctorat, Polytechnique Montréal]. PolyPublie.
Citation: <https://publications.polymtl.ca/54854/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/54854/>
PolyPublie URL:

Directeurs de recherche: Louis-Martin Rousseau, & Quentin Cappart
Advisors:

Programme: Doctorat en mathématiques
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Apprentissage Profond en Optimisation Combinatoire : Apprentissage de
Bornes et Résolution de Problèmes de Transport**

AUGUSTIN PARJADIS DE LARIVIÈRE

Département de mathématiques et de génie industriel

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

Août 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Apprentissage Profond en Optimisation Combinatoire : Apprentissage de
Bornes et Résolution de Problèmes de Transport**

présentée par **Augustin PARJADIS DE LARIVIÈRE**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*

a été dûment acceptée par le jury d'examen constitué de :

Thibaut VIDAL, président

Louis-Martin ROUSSEAU, membre et directeur de recherche

Quentin CAPPART, membre et codirecteur de recherche

Jorge MENDOZA, membre

Jean-Charles RÉGIN, membre externe

DÉDICACE

À Papa et Léopold.

"J'ai toujours aimé le désert. On s'assoit sur une dune de sable. On ne voit rien. On n'entend rien. Et cependant quelque chose rayonne en silence..."

-Antoine de Saint-Exupéry

REMERCIEMENTS

Je tiens à remercier les personnes qui ont joué un rôle crucial dans la réalisation fructueuse de ce projet de doctorat, ainsi que celles qui ont contribué par leur présence à rendre cette expérience particulièrement plaisante et enrichissante.

Je voudrais avant tout exprimer ma profonde gratitude envers mon directeur de recherche, Professeur Louis-Martin Rousseau, dont l’expertise immense et les idées lumineuses et incessantes furent une source précieuse d’inspiration et de motivation. En plus d’avoir supervisé ma maîtrise à mon arrivée au Canada, le Professeur Rousseau m’a donné l’occasion de diriger des séances de laboratoire en recherche opérationnelle, une expérience extrêmement enrichissante et formatrice. Je tiens également à remercier mon co-directeur Professeur Quentin Cappart pour sa supervision, son engagement et ses suggestions toujours judicieuses. Professeur Cappart m’a accompagné tout au long de ma maîtrise et de mon doctorat, et je lui dois beaucoup pour les compétences que j’ai acquises tout au long de mon parcours.

Merci à Emmanuel Rachelson et son dynamisme dans les projets étudiants peu simples à lancer ; nous n’avons finalement rien publié sur Deep TD(λ) mais tu m’as donné goût à la recherche et fus une source d’inspiration dans l’étude de l’optimisation et de l’apprentissage profond. ISAE-Supaéro et sa filière de sciences de la décision te doivent beaucoup.

Bien que la COVID ait bouleversé la vie de laboratoire, je tiens à chaleureusement remercier mes collègues de bureau et de travail, au CIRRELT, au Pavillon André-Aisenstadt et à la Poly, mais particulièrement Isaac, Peyman, Mahdis et Adrien dont la présence et le soutien furent précieux en recherche et en amitié. Merci Eva pour ton soutien quotidien et tes inspirations de problèmes d’optimisation tirés de la mécanique et l’énergie hydroélectrique !

J’ai eu la chance de profiter de multiples opportunités de stages : merci à l’équipe de Fastercom et à Romain Bonnifet pour leur accueil, ainsi qu’à Quentin Massoteau pour ses conseils et sa précieuse expérience. Je tiens également à remercier toute l’équipe de recherche dynamique et innovante avec qui j’ai eu la chance de travailler chez ElementAI, Nicolas Chapados, Chief Science Officer, mais aussi Raymond, Nicolas, Cyril et Isabeau avec qui j’ai beaucoup appris et passé d’excellents moments.

Je remercie du fond du cœur mes parents dont le soutien et les encouragements dans la poursuite de mes études furent indéfectibles. À mon père, Géraud, qui fut une source d’inspiration continue, un interlocuteur sans égal, un formateur idéal, un guide passionné et un père fier qui veille maintenant depuis le royaume céleste.

RÉSUMÉ

Nous assistons récemment à une montée en puissance de l'importance de l'apprentissage automatique et particulièrement de l'apprentissage profond dans de nombreux domaines. Si l'optimisation est indispensable à l'entraînement des réseaux de neurones profonds, des applications de l'apprentissage profond font à leur tour apparition en optimisation mathématique afin d'en enrichir les possibilités. Cette thèse se concentre sur l'application des techniques d'apprentissage profond pour l'amélioration d'approches classiques de recherche opérationnelle dans différents domaines.

Le premier projet propose une méthode d'intégration de bornes d'optimisation apprises à l'intérieur d'un solveur branch-and-bound, en utilisant une combinaison d'apprentissage par renforcement et de diagrammes de décision. Les difficultés d'une association d'un solveur classique et performant à des inférences de réseaux de neurones profonds sont présentées et des solutions sont développées et testées afin de résoudre jusqu'à 15% plus rapidement des problèmes de stable maximum.

Le deuxième projet aborde le problème de la gestion d'une flotte de véhicules sous incertitude, en utilisant une méthodologie basée sur de l'apprentissage supervisé qui s'adapte à l'activité de la flotte, à la zone géographique et à la variation des demandes quotidiennes. Une méthode simple d'apprentissage au service de répartiteurs est présentée et permet une amélioration de plus de 10% de la réactivité de la flotte de l'entreprise partenaire. Le succès de ce projet a donné lieu à une collaboration prolongée et à une nouvelle étude d'intégration d'apprentissage automatique pour la prise de rendez-vous en temps réel évitant les résolutions répétées d'un problème de tournée de véhicule complexe.

Le dernier projet propose une approche d'apprentissage profond non-supervisé pour la génération de facteurs lagrangiens utilisés dans la relaxation de Held et Karp du problème très général de voyageur de commerce, afin de générer rapidement des bornes performantes dans un solveur dédié. L'intérêt de l'apprentissage machine dans la résolution d'un problème de voyageur de commerce en exploitant la structure de graphe est testée et démontrée, aboutissant à des réductions du temps de résolution de l'ordre de 20%. L'approche développée est suffisamment générale pour être transposable à d'autres relaxations lagrangiennes.

Les apports de l'apprentissage profond dans le domaine de l'optimisation explorés dans cette thèse se manifestent principalement à travers deux types de contributions : d'une part, la prédiction de valeurs inconnues permet d'intégrer directement les informations tirées de données passées dans un problème d'optimisation dont le résultat sera plus robuste et adapté ; d'autre

part, les algorithmes d'optimisation développent des techniques avancées et sophistiquées afin de gagner en performance dans un domaine où les problèmes sont généralement $\mathcal{NP}$ -difficiles et les gains de plus en plus marginaux, mais l'apprentissage automatique permet de découvrir ou remplacer des heuristiques en s'adaptant notamment aux problèmes rencontrés. Ces projets démontrent le potentiel de l'apprentissage profond dans des problèmes d'optimisation mathématique complexes, et soulignent l'importance de l'incorporation des techniques d'apprentissage dans le domaine de la recherche opérationnelle. Des méthodes d'apprentissage supervisé, non-supervisé et par renforcement sont explorés et utilisés en fonction de la nature de problèmes rencontrés, mettant en évidence la richesse des approches possibles.

ABSTRACT

We are witnessing a recent surge in the importance of machine learning, particularly deep learning, in various fields. While optimization is essential for training deep neural networks, applications of deep learning are also emerging in mathematical optimization to enrich its capabilities. This thesis focuses on the application of deep learning techniques to enhance classical approaches in operations research across different domains.

The first project proposes a method for integrating learned optimization bounds within a branch-and-bound solver using a combination of reinforcement learning and decision diagrams. The challenges of combining a traditional and efficient solver with deep neural network inferences are presented, and solutions are developed and tested to solve maximum independent set problems up to 15% more quickly.

The second project addresses the problem of managing a fleet of vehicles under uncertainty, using a supervised learning pipeline that adapts to fleet activity, geographic zones, and daily demand variations. A simple learning pipeline for dispatchers is presented, resulting in over 10% improvement in the fleet’s responsiveness for the partner company. The success of this project has led to extended collaboration and a new study on integrating machine learning for real-time appointment scheduling, avoiding repeated resolutions of complex vehicle routing problems.

The third project proposes an unsupervised deep learning approach for generating Lagrangian factors used in the Held-Karp relaxation of the traveling salesperson problem, enabling the generation of high-quality bounds in a dedicated solver. The usefulness of machine learning in solving the traveling salesperson problem by exploiting graph structure is tested and demonstrated, with gains of up to 20% on solving time. The approach developed is general enough to be applicable to other Lagrangian relaxations.

On one hand, predicting unknown values allows for the direct integration of information derived from past data into an optimization problem, resulting in more robust and adapted outcomes. On the other hand, optimization algorithms develop advanced and sophisticated techniques to enhance performance in a domain where problems are typically $\mathcal{NP}$ -hard and marginal gains are increasingly challenging. However, machine learning enables the discovery or replacement of heuristics by adapting to the encountered problems.

These projects demonstrate the potential of deep learning in complex mathematical optimization problems and underscore the importance of incorporating learning techniques in

the field of operations research. Supervised, unsupervised, and reinforcement learning methods are explored and utilized based on the nature of the encountered problems, highlighting the richness of possible approaches.

TABLE DES MATIÈRES

DÉDICACE	iii
REMERCIEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vii
TABLE DES MATIÈRES	ix
LISTE DES TABLEAUX	xiii
LISTE DES FIGURES	xiv
LISTE DES SIGLES ET ABRÉVIATIONS	xvi
LISTE DES ANNEXES	xvii
CHAPITRE 1 INTRODUCTION	1
1.1 Motivation	1
1.2 Contexte	2
1.3 Objectifs de recherche	3
CHAPITRE 2 REVUE DE LITTÉRATURE	5
2.1 Recherche opérationnelle	5
2.1.1 Branch-and-bound	6
2.1.2 Diagrammes de décision	6
2.2 Apprentissage profond	8
2.2.1 Fonction de perte et rétropropagation de gradient	9
2.2.2 Apprentissage par renforcement	10
2.2.3 Réseau de neurones sur graphes	11
2.2.4 Mécanisme d'attention	12
2.3 Hybridation de l'apprentissage et de l'optimisation	12
CHAPITRE 3 DÉMARCHE ET ORGANISATION DE LA THÈSE	14

CHAPITRE 4	ARTICLE 1 : IMPROVING BRANCH-AND-BOUND USING DECISION DIAGRAMS AND REINFORCEMENT LEARNING	17
4.1	Introduction	17
4.2	Learning Bounds Inside Branch-and-Bound	18
4.2.1	Decision Diagram-Based Branch-and-Bound	18
4.2.2	Variable Ordering and Reinforcement Learning	19
4.2.3	The Branch-and-Bound Algorithm with a RL Agent	20
4.3	Experimental Results	22
4.3.1	Performances of the Learned Variable Ordering	22
4.3.2	Caching to save computation time	23
4.3.3	Discussion	24
4.4	Conclusion	24
CHAPITRE 5	ARTICLE 2 : REPOSITIONING FLEET VEHICLES : A LEARNING PIPELINE	26
5.1	Introduction	26
5.2	Literature review	27
5.3	The Ride-hailing Problem	28
5.3.1	Problem Definition	28
5.3.2	Modelling the Repositioning Task	30
5.4	Learning Pipeline for Vehicle Repositioning	30
5.5	Case Study : Ambulance Fleet in Belgium	34
5.5.1	Use of the Pipeline	35
5.5.2	Training Phase	35
5.5.3	Results : Location Prediction	36
5.5.4	Results : Ambulances Repositioning	37
5.6	Conclusion	39
CHAPITRE 6	APPRENTISSAGE AUTOMATIQUE POUR L'APPROXI- MATION ET LA RÉOLUTION DE PROBLÈMES DE PLANIFICA- TION DE RENDEZ-VOUS DANS DES TOURNÉES DE VÉHICULES EN TEMPS RÉEL	40
6.1	Introduction	40
6.1.1	Motivation	40
6.1.2	Préliminaires	40
6.1.3	Approche	42

6.2	Représentation d'une solution au VRP	42
6.2.1	Structure du graphe	43
6.2.2	Vecteurs paramètre	43
6.3	Apprentissage sur des solutions de VRP	44
6.3.1	Score d'insertion	45
6.3.2	Ensemble d'entraînement	45
6.3.3	Apprentissage	46
6.4	Application aux tournées de techniciens	47
6.5	Conclusion	48
CHAPITRE 7 RELAXATION DE HELD ET KARP ET VOYAGEUR DE COMMERCE : GÉNÉRATION DE BORNE DUALE PAR APPRENTIS- SAGE PROFOND NON-SUPERVISÉ		49
7.1	Introduction	49
7.1.1	Motivation	49
7.1.2	Revue de littérature	49
7.1.3	Approche	50
7.2	Relaxation lagrangienne de Held et Karp	50
7.3	Apprentissage non-supervisé pour Held et Karp	53
7.3.1	Génération des facteurs lagrangiens	53
7.3.2	Dérivation pour la rétropropagation de gradient	53
7.3.3	Génération des données	54
7.4	Application	55
7.4.1	Entraînement	55
7.4.2	Intégration branch-and-bound	56
7.5	Conclusion	58
CHAPITRE 8 DISCUSSION GÉNÉRALE		60
8.1	Apprentissage automatique et faisabilité	60
8.2	Modélisation pour l'apprentissage et l'optimisation	61
8.3	Traitement des données	61
CHAPITRE 9 CONCLUSION ET RECOMMANDATIONS		63
9.1	Synthèse des travaux	63
9.2	Limitations et améliorations futures	64
RÉFÉRENCES		66

ANNEXES	78
-------------------	----

LISTE DES TABLEAUX

Table 4.1	Average time and nodes explored for 100 MISP instances of size n .	23
Table 5.1	State features	32
Table 5.2	Results of the different predictors for the next request. The percentage of improvement compared to the most frequent heuristic is also proposed.	36
Table 5.3	Distance (km) to drive to requests in 10 simulated days.	38
Table 5.4	Distance (km) to drive to requests, averaged over 20 days.	38
Table 6.1	Précision de l'insertion.	48
Table 7.1	Différentes générations de facteurs lagrangiens pour la relaxation de Held et Karp au sein d'un algorithme de branch-and-bound ($n = 50$).	57
Table 7.2	Différentes générations de facteurs lagrangiens pour la relaxation de Held et Karp au sein d'un algorithme de branch-and-bound ($n = 80$).	58
Table 7.3	Différentes générations de facteurs lagrangiens pour la relaxation de Held et Karp au sein d'un algorithme de branch-and-bound ($n = 100$).	58
Table A.1	Average number of nodes, optimality gap, and execution time for solving 100 MISP instances of size n . Results show the superiority of DD-RL+ compared to CPLEX and other DD methods for the three configurations.	81
Table B.1	Accuracy of the different predictors for the next request.	83
Table C.1	Précision des différents prédicteurs	87
Table C.2	Distances parcourues sur dix journées simulées	88
Table C.3	Score de simulation par rapport aux participants du concours DIMACS	89
Table C.4	Distances (km) parcourues sur 10 jours simulés.	91

LISTE DES FIGURES

Figure 2.1	Restriction d'un diagramme de décision	8
Figure 2.2	Relaxation d'un diagramme de décision	8
Figure 4.1	Performance profile of a GNN agent and MinState for variable ordering over 100 instances of the MISP	24
Figure 4.2	Performance profile of an improved GNN agent and MinState for variable ordering over 100 instances of the MISP	25
Figure 5.1	Ride-hailing situation - example.	29
Figure 5.2	Pipeline structure.	31
Figure 5.3	Statistics for the training phase.	36
Figure 5.4	Distance to drive to requests in 10 simulated days (data in Table 5.3)	37
Figure 6.1	Exemple de graphe, illustrant des arêtes E_{A-R} , E_{A-T} , E_{R-T} , et E_{T-T} .	43
Figure 6.2	Exemple de graphe, illustrant des distances sur arêtes E_{A-R} , et E_{A-A} .	44
Figure 6.3	Courbes d'erreur quadratique moyenne $(s - s_{a,\tau})^2$ sur l'ensemble d'entraînement et de validation.	47
Figure 7.1	Relaxation de Held et Karp sans (1) et avec (2) pénalités.	52
Figure 7.2	Entraînement pour la génération de facteurs lagrangiens sur des graphes G_1^{50}	56
Figure 7.3	Comparaison du nombre de noeuds et du temps de résolution du TSP sur des graphes de taille 50 entre les relaxations HK et GNN_{10}	57
Figure 7.4	Comparaison du nombre de noeuds et du temps de résolution du TSP sur des graphes de taille 50 entre les relaxations HK et GNN_{10} (points extrêmes retirés).	57
Figure 7.5	Comparaison du nombre de noeuds et du temps de résolution du TSP sur des graphes de taille 80 entre les relaxations HK et GNN_{10}	58
Figure 7.6	Comparaison du nombre de noeuds et du temps de résolution du TSP sur des graphes de taille 100 entre les relaxations HK et GNN_{10} . . .	59

Figure A.1	Branch-and-bound results for 100 MISP instances (Barabasi-Albert graphs of 140 nodes, with $\nu = 4$). The y -axis represents the percentage of instances that are able to solve the problem at optimality, within a given number of nodes explored (x -axis). These results show that the our learned solver without improvement (DD-RL) is able to reduce the number of nodes explored compared to the commonly used heuristic (MIN), but at the expense of a significantly longer execution time. The improved version of the solver (DD-RL+) is able to compensate for this limitation.	80
Figure B.1	Features importance.	82
Figure B.2	Retraining accuracy	84
Figure C.1	Score d'entraînement pour un agent d'apprentissage par renforcement pour l'attribution	89

LISTE DES SIGLES ET ABRÉVIATIONS

GNN	Graph Neural Network (réseau de neurones sur graphes)
TSP	Travelling Salesman Problem (problème du voyageur de commerce)
VRP	Vehicle Routing Problem (problème de tournée de véhicules)
DD	Decision Diagrams (diagrammes de décision)
MISP	Maximum Independent Set Problem (problème du stable maximum)
BnB	Branch-and-Bound (algorithme de séparation et évaluation)
NN	Neural Network (réseau de neurones)
MLP	Multilayer Perceptron (perceptron multicouche)
RL	Reinforcement Learning (apprentissage par renforcement)
CO	Combinatorial Optimization (optimisation combinatoire)
MSE	Mean Squared Error (erreur quadratique moyenne)
EGAT	Edge-Featured Graph Attention Network (réseau de neurones sur graphes avec mécanisme d'attention et paramètre sur les arêtes)
MIP	Mixed Integer Programming (programmation mixte en nombres entiers)

LISTE DES ANNEXES

Annexe A	SUPPLÉMENTS DU CHAPITRE 4	78
Annexe B	ANNEXES - <i>Repositioning Fleet Vehicles : a Learning Pipeline</i> . . .	82
Annexe C	SUPPLÉMENTS DU CHAPITRE 5	85

CHAPITRE 1 INTRODUCTION

1.1 Motivation

De nombreux problèmes de prise de décision peuvent être représentés par un modèle mathématique, et spécifiquement comme un problème d'optimisation sous contraintes. La recherche opérationnelle est une discipline qui utilise des méthodes analytiques et mathématiques pour modéliser, analyser et résoudre des problèmes complexes liés à la prise de décision. Elle offre des outils et des techniques pour représenter de manière formelle les problèmes, explorer différentes solutions et trouver les meilleures options en fonction des objectifs spécifiques. Opérationnellement, la fonction objectif à minimiser ou maximiser représente le plus souvent une mesure de coût ou de gain, et les contraintes représentent les restrictions imposées aux décisions à prendre. Une modélisation en programme linéaire en nombres entiers est courante mais en théorie difficile à résoudre car c'est un problème $\mathcal{NP}$ -difficile, c'est à dire qu'on ne connaît pas de méthode générale pour le résoudre en temps polynomial. En pratique, des solveurs performants existent et permettent de résoudre des problèmes de grande taille ; des applications fructueuses et courantes incluent la planification de la production ou de la distribution d'énergie, la gestion de la chaîne d'approvisionnement, l'ordonnancement de tâches, l'affectation des ressources, l'optimisation de portefeuille, etc. Ces problèmes sont très étudiés, et leurs nombreuses applications concrètes motivent la recherche depuis des décennies.

D'un autre côté, l'apprentissage automatique est une approche prédictive, basée plus spécifiquement sur l'utilisation et l'exploitation de données moins structurées, et les possibilités ouvertes par l'apprentissage automatique se démultiplient grâce à l'apprentissage profond. Celui-ci devient incontournable pour la reconnaissance d'images, la traduction automatique, le traitement du langage naturel, la recommandation de produits, la détection de fraudes, la reconnaissance vocale, etc.

Cette thèse s'intéresse à l'interaction entre ces deux disciplines. Si les solveurs d'optimisation sont performants sur les problèmes réels pour lesquels on les utilise, c'est souvent grâce au travail d'adaptation des méthodes de recherche aux problèmes en question grâce à l'expertise humaine ; l'apprentissage automatique peut ainsi présenter une opportunité d'adaptation automatique des algorithmes de recherche opérationnelle aux problèmes rencontrés, et ce de manière autonome ou complémentaire à l'expertise humaine.

1.2 Contexte

L'optimisation mathématique englobe différents concepts et techniques utilisés pour résoudre des problèmes d'optimisation. L'optimisation linéaire est l'un de ces concepts clés, où l'objectif est de maximiser ou minimiser une fonction linéaire sous des contraintes également linéaires. Ce type de problème est couramment résolu par exemple pour la planification de production, l'affectation de ressources et la gestion de stocks.

L'optimisation combinatoire est un autre domaine important, pour lequel la nature discrète des variables complexifie les méthodes de résolution. Les problèmes du voyageur de commerce, de l'affectation des tâches et de l'ordonnancement sont des exemples classiques d'optimisation combinatoire. Résoudre ces problèmes nécessite des techniques spécifiques telles que la recherche locale, les méthodes heuristiques et les algorithmes de branchement et de coupes.

Pour résoudre efficacement ces problèmes d'optimisation, différents solveurs et outils logiciels ont été développés. Les plus connus incluent CPLEX, Gurobi, SCIP, Gecode et GLPK, et exploitent des techniques avancées telles que la programmation linéaire en nombres entiers, la programmation par contraintes et la programmation non linéaire pour trouver des solutions optimales ou proches de l'optimum.

Cependant, les méthodes traditionnelles ne permettent pas toujours de trouver des solutions optimales pour des problèmes complexes, et des métaheuristiques sont alors souvent utilisées. Ces techniques d'optimisation se concentrent sur la recherche heuristique et l'exploration locale, et incluent des algorithmes évolutionnaires, des algorithmes de recherche tabou, des algorithmes génétiques, etc. Ces méthodes peuvent fournir des solutions de qualité acceptable dans des délais raisonnables, mais sans garantie d'optimalité.

La résolution de problèmes d'optimisation peut être complexe en termes de temps de calcul, la taille du problème et le nombre de contraintes influençant la complexité de la résolution ; on appelle explosion combinatoire la croissance exponentielle du nombre de solutions possibles à mesure que la taille d'un problème augmente. Certains problèmes sont connus pour être NP-difficiles, signifiant qu'il n'existe pas d'algorithme efficace pour les résoudre exactement dans un temps polynomial. Dans ces cas, des compromis doivent être faits entre la qualité de la solution et le temps de calcul. La complexité des problèmes d'optimisation et la recherche de solutions optimales restent des défis importants dans ce domaine.

L'application de l'optimisation mathématique aux problèmes réels de recherche opérationnelle présente plusieurs défis, notamment la gestion des informations manquantes ou imparfaites. Il est souvent difficile d'obtenir des données complètes et précises pour modéliser le problème de manière juste, ce qui peut affecter la qualité des solutions obtenues. De plus, la nécessité de

prendre des décisions en temps réel va souvent à l’encontre des objectifs d’optimalité souvent recherchés en optimisation mathématique.

1.3 Objectifs de recherche

Si l’optimisation est essentielle à l’entraînement de modèles d’apprentissage, tout particulièrement les réseaux de neurones profonds entraînés par descente de gradient stochastique, l’apprentissage machine peut également ouvrir de nouvelles possibilités en optimisation mathématique. Cette thèse a pour objectif général de proposer des utilisations avantageuses de l’apprentissage automatique pour la recherche opérationnelle, particulièrement pour la découverte d’heuristiques avec des agents génératifs ou de prise de décisions séquentielles, ou pour la prédiction de paramètres inconnus. Si l’apprentissage automatique et la recherche opérationnelle semblent être deux approches différentes pour la prise de décision, elles ont chacune leurs avantages propres, leurs inconvénients, et ne sont pas nécessairement exclusives mais peuvent au contraire s’enrichir l’une l’autre. Nous explorons au sein de différents projets cet enrichissement mutuel, tout particulièrement les apports de l’apprentissage à l’optimisation mathématique, et en tirons des applications pratiques pour la résolution de problèmes complexes.

Les Chapitres 4 et 7 proposent plusieurs approches pour la question de l’intégration de bornes apprises dans un algorithme de branch-and-bound. En effet, on se demande s’il est possible d’accélérer une résolution de problème en rajoutant des inférences de modèle d’apprentissage profond à chaque étape de la recherche pour la construction de diagrammes de décision. Une approche naïve au Chapitre 4 montre que le coût de l’inférence est trop important et d’autres approches sont explorées pour effectivement accélérer la résolution de problèmes de stable maximum (*maximum independent set*) grâce aux bornes apprises. Cette approche sera complétée par l’apprentissage non-supervisé de bornes grâce à la relaxation lagrangienne de Held et Karp au Chapitre 7, également intégrée dans un algorithme de branch-and-bound pour le voyageur de commerce.

Les Chapitres 5 et 6 se demandent comment utiliser les données passées de problèmes concrets de recherche opérationnelle pour améliorer les services proposés dans le futur. Ainsi, le Chapitre 5 explore comment les données de transport ambulancier peuvent être utilisées pour mieux repositionner des ambulances d’une flotte en vue d’en améliorer le temps de réponse. Le Chapitre 6 se penche sur l’utilisation de solutions de tournées de véhicules pour des techniciens ou infirmières à domicile afin de prédire quelles insertions de rendez-vous sont possibles et pertinentes sans résolutions coûteuses immédiates de nouveaux problèmes de tournées. Les outils développés lors de ces projets font l’objet d’un déploiement opérationnel avec des

partenaires industriels.

CHAPITRE 2 REVUE DE LITTÉRATURE

Ce chapitre présente une courte introduction aux différents domaines et champs de recherche qui ont mené aux contributions qui suivent grâce une revue de littérature.

2.1 Recherche opérationnelle

Née pendant la seconde guerre mondiale puis formalisée comme la spécialité interdisciplinaire moderne de la direction et gestion de systèmes complexes, la recherche opérationnelle est une science de la décision tirant essentiellement parti de développements mathématiques en résolution de problèmes complexes. La partie mathématique consiste en la modélisation mathématique des problèmes afin de les résoudre grâce à un solveur. L'approche la plus simple est celle des modèles linéaires, pour lesquels la fonction objectif et les contraintes font intervenir les variables de décision linéairement, qui sont facilement résolus grâce à l'algorithme du simplexe [1,2] ou des points intérieurs [3] et permettent de modéliser de nombreux problèmes. Un paradigme de modélisation plus expressif est celui de la programmation linéaire en nombres entiers, dont le domaines de définition discrets compliquent la résolution ; des algorithmes branchant sur les valeurs fractionnaires de la relaxation linéaire (*branch-and-bound* [4, 5]) ou ajoutant également des coupes pour resserrer cette même relaxation (*branch-and-cut* [6]) sont alors par exemple utilisés. De nombreuses approches existent modéliser la diversité des problèmes possibles, comme la programmation non-linéaire qui s'affranchit des contraintes de linéarité, ou la programmation stochastique prenant en compte variables aléatoires ou paramètres incertains. Si la résolution de grands problèmes est facilitée par l'explosion de la puissance de calcul des ordinateurs, avec jusqu'à récemment une quantité de transistors présents sur une puce de microprocesseur doublant tous les deux ans [7, 8], elle est aussi possible grâce à de nombreuses innovations en optimisation. Les méthodes de décomposition permettent par exemple de partager le problème en un problème maître et des sous-problèmes qui sont résolus itérativement jusqu'à convergence avec ajout à chaque étape de lignes (ajout de contraintes, décomposition de Benders [9]) ou colonnes (ajout de variables, décomposition de Dantzig-Wolfe [10, 11]) au problème maître. Cette diversité mène à une pléthore de techniques de résolution et de solveurs paramétrés les combinant. Une des applications les plus célèbres de l'optimisation mathématique est celle du problème combinatoire du voyageur de commerce [12, 13] qui pose les bases de nombreux problèmes industriels faisant intervenir des tournées de véhicules ou des plus courts chemins.

2.1.1 Branch-and-bound

Si des algorithmes de résolution efficaces existent pour la programmation linéaire (simplexe, points intérieurs), la programmation linéaire en nombres entiers demande une nouvelle approche. On peut s'intéresser au problème en nombres entiers sans la contrainte d'entièreté des variables : on appelle ce problème la relaxation linéaire du problème original, et cette relaxation peut être résolue comme un problème linéaire classique afin d'obtenir une borne duale sur le problème original. Si la solution obtenue est entière, le problème original est résolu à optimalité et l'on peut s'arrêter là ; sinon, on peut choisir une variable fractionnaire pour partager l'espace de solution en deux autour de cette variable, en imposant qu'elle soit inférieure ou égale et supérieure ou égale aux entiers encadrant la valeur fractionnaire. Cette idée de séparation et évaluation, communément appelée *branch-and-bound*, donne lieu à un algorithme de résolution qui produit un arbre de recherche, dans lequel les noeuds sont des sous-problèmes du problème original, grâce à la démarche décrite ci-dessus. En considérant un problème de minimisation : (1) un noeud non exploré de l'arbre est sélectionné, (2) des sous-problèmes sont générés en divisant le noeud actuel en plusieurs branches, (3) une borne inférieure est générée pour chaque sous-problème, (4) la branche est élaguée si la borne inférieure est supérieure à la borne supérieure actuelle, sinon l'exploration continue, et la borne supérieure est mise à jour si une meilleure solution est trouvée. L'utilisation des bornes d'optimisation permet de ne pas énumérer et explorer tous l'arbre de recherche, ce qui serait impossible pour de grands problèmes. Cet algorithme de branch-and-bound est largement utilisé pour la résolution de problèmes linéaires en nombres entiers dans de nombreux solveurs (Gurobi [14], CPLEX [15], etc.) et par son ubiquité fait l'objet de nombreuses études et extensions [16, 17]. Notamment, de nombreux choix heuristiques sont faits pendant l'exploration branch-and-bound et constituent alors des possibilités d'améliorations immédiates, créant ainsi toute une littérature étudiant par exemple le choix des variables de branchement ou des noeuds à explorer [18, 19].

2.1.2 Diagrammes de décision

Les diagrammes de décision sont un outil de représentation graphique compacte de l'ensemble des solutions de problèmes d'optimisation discrète. Ce sont des multigraphes dirigés acycliques, comportant un noeud racine et un noeud terminal, organisés en couches, et dont les arcs représentent des assignations de variables. Sommairement, ils peuvent être vus comme des arbres de décisions dont les noeuds représentant des états équivalents sont fusionnés. Ils sont historiquement utilisés pour la représentation de fonction booléennes et la conception et vérification de circuits [20, 21], et sont récemment utilisés pour de nombreuses autres ap-

plications en optimisation [22, 23] par exemple pour des problèmes d’ordonnancement ou de *bin packing* multi-dimensionnels [24, 25], ou pour représenter des contraintes globales ou travailler avec des variables stochastiques [26, 27]. Ils sont en pratique beaucoup utilisés pour la génération de bornes d’optimisation [28], comme une alternative à la relaxation linéaire pour les bornes duales, et peuvent être utilisés pour définir un algorithme de branch-and-bound exploitant leur structure [29–32].

Pour un problème d’optimisation discrète, chaque solution est représentée dans un diagramme de décision par un chemin du noeud racine au noeud terminal, avec chaque arc emprunté assignant une valeur à une variable ; de la même manière, chaque chemin du noeud racine au noeud terminal représente une solution réalisable pour le problème. Résoudre le problème d’optimisation revient donc alors à trouver le plus court chemin dans le diagramme de décision, si l’on s’intéresse à un problème de minimisation, le plus long pour une maximisation. En pratique, les diagrammes de décision sont trop lourds à construire, stocker et exploiter pour de gros problèmes et l’on préfère donc construire des diagrammes restreints ou relaxés qui vont représenter un sous-ensemble ou un sur-ensemble de l’espace des solutions. Un diagramme de décision restreint limite le nombre de noeuds par couche, et retire ainsi les noeuds excédentaires, ainsi que leurs arcs, pour mettre en oeuvre cette limitation ; Figure 2.1 illustre une restriction de largeur 2 pour un diagramme de décision à 5 variables. Cette procédure fait perdre de l’information, et donc de potentielles solutions, ce qui en fait une représentation d’un sous-ensemble des solutions. La résolution de ce diagramme de décision restreint, d’une largeur et donc d’une complexité arbitraire, permet alors la génération d’une borne primale sur le problème, toutes les solutions comprises dans le diagramme de décision restreint étant réalisables. De la même manière, un diagramme de décision relaxé limite le nombre de noeuds par couche en fusionnant des noeuds, et leurs arcs, sur les couches pour lesquelles ils sont trop nombreux. Cette procédure, illustrée Figure 2.2, garde toute l’information présente mais fait apparaître des chemins correspondant à des solutions non-réalisables : le diagramme encode ainsi un sur-ensemble des solutions et permet de générer une borne duale, dont la qualité dépendra de la largeur maximale choisie.

Les diagrammes restreints et relaxés peuvent être utilisés au sein d’un algorithme de branch-and-bound : on peut sélectionner des noeuds exacts (non restreints ou relaxés) couvrant la largeur du diagramme et développer de nouveaux diagrammes prenant ces noeuds pour racines qui sont à leur tour restreints/relaxés, etc. Cet algorithme permet essentiellement de brancher sur des solutions partielles du problème. De nombreux choix heuristiques rentrent en jeu [29, 33] (choix des noeuds à retirer en restriction, à fusionner en relaxation, sur lesquels brancher, choix de l’ordre des variables pour les couches du diagramme, etc.) et peuvent ainsi être sources de gains potentiels.



FIGURE 2.1 Restriction d'un diagramme de décision



FIGURE 2.2 Relaxation d'un diagramme de décision

2.2 Apprentissage profond

L'apprentissage profond est une branche de l'apprentissage automatique utilisant spécifiquement des réseaux de neurones artificiels comme approximateurs de fonction universels [34]. Les perceptrons multicouches, introduits en 1962 par Frank Rosenblatt, réalisent leur potentiel applicatif récemment grâce à la plus grande disponibilité de quantité massives de données et de puissance de calcul ; on cite notamment des applications liées à la classification d'images [35, 36], le traitement automatique du langage [37, 38] ou la conception de médicament [39, 40]. Différents types d'apprentissage existent, de l'apprentissage supervisé qui apprend à partir d'exemples étiquetés, à l'apprentissage non-supervisé utilisé pour extraire des structures et des modèles à partir de données non étiquetées, ou encore l'apprentissage par renforcement dans lequel un agent autonome apprend à agir avec son environnement

et à effectuer une tâche par essais et erreurs. Nous renvoyons le lecteur intéressé vers une introduction complète sur ce sujet [41] et développons ci-dessous les points importants qui seront utilisés à travers cette thèse.

2.2.1 Fonction de perte et rétropropagation de gradient

La fonction de perte est une mesure de l'écart entre les prédictions faites par un modèle et les valeurs réelles des données. Elle évalue la performance du modèle en quantifiant à quel point les prédictions correspondent aux véritables étiquettes ou valeurs cibles. L'objectif est de minimiser cette fonction de perte pour obtenir des prédictions précises, le plus proche possible de la référence. Différentes tâches d'apprentissage, comme la classification ou la régression, utilisent différentes fonctions de perte adaptées à leur nature. Par exemple, la fonction de perte couramment utilisée dans la classification binaire est l'entropie croisée (*cross-entropy*, Équation 2.1), tandis que la régression linéaire utilise souvent l'erreur quadratique moyenne (*mean squared error*, Équation 2.2); on note y l'exemple cible et $\hat{y}$ la prédiction. D'autres fonctions de pertes peuvent être utilisées en fonction de l'application ou des performance recherchées, comme la *dice loss* en imagerie médicale [42] ou la *Wasserstein loss* pour les modèles génératifs [43]; nous renvoyons le lecteur vers Wang et al. [44] pour une étude détaillée.

$$\text{CrossEntropyLoss}(y, \hat{y}) = - \sum_{i=1}^n y_i \log(\hat{y}_i) \quad (2.1)$$

$$\text{MSE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (2.2)$$

Une fois la fonction de perte définie, l'algorithme de rétropropagation de gradient est utilisé pour ajuster les paramètres du modèle pour minimiser la perte. Cette technique d'optimisation calcule le gradient des paramètres par rapport à la fonction de perte, puis met à jour les paramètres dans la direction opposée du gradient pour minimiser la perte. En plusieurs étapes : (1) les données d'entrée sont propagées à travers le modèle pour obtenir des prédictions; chaque couche du modèle effectue des transformations des données en utilisant ses paramètres, (2) les prédictions sont comparées aux valeurs réelles à l'aide de la fonction de perte, qui mesure l'écart entre les prédictions et les cibles, (3) les gradients de la fonction de perte par rapport aux paramètres du modèle sont calculés à l'aide de la règle de dérivation des fonctions composées, et (4) les paramètres du modèle sont mis à jour en utilisant une méthode d'optimisation qui ajuste les paramètres dans la direction opposée du gradient pour minimiser la perte. Ce processus de rétropropagation est répété itérativement jusqu'à ce que

la fonction de perte soit suffisamment réduite ou que le modèle converge vers une solution satisfaisante. Bien que l'on puisse utiliser une simple descente de gradient stochastique, des algorithmes d'optimisation de réseaux de neurones plus performants existent [45].

2.2.2 Apprentissage par renforcement

L'apprentissage par renforcement est fondamentalement différent de l'apprentissage supervisé par sa nature séquentielle et l'absence de modèle à reproduire ; plutôt que de créer un prédicteur cherchant à reproduire des données de référence, l'apprentissage par renforcement cherche à créer des agents capables d'apprendre à prendre des décisions et à agir dans un environnement pour maximiser une récompense (ou minimiser une pénalité). Un environnement défini pour l'apprentissage par renforcement comprend à tout moment un ensemble d'états observables $\mathcal{S}$ et un ensemble d'actions possibles $\mathcal{A}$. À chaque étape, l'agent observe l'état de l'environnement, choisit une action à effectuer et reçoit une récompense ou une pénalité en fonction de cette action. L'objectif de l'agent est de trouver la meilleure politique, soit la meilleure stratégie d'action ou fonction $\pi : \mathcal{S} \rightarrow \mathcal{A}$ reliant une action à un état, qui maximise la somme des récompenses sur la durée de son interaction avec l'environnement. Cette politique peut être dérivée de la fonction de valeur actions-états $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (*q-values*) associant le gain espéré $Q(s, a)$ en effectuant l'action a dans l'état s . L'algorithme de *Q-learning* [46] permet par exemple d'estimer cette fonction par l'expérience grâce aux ajustements itératifs

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a)) \quad (2.3)$$

où $\alpha \in]0, 1]$ est un taux d'apprentissage, r_t la récompense reçue entre t et $t + 1$, $\gamma \in [0, 1]$ un facteur d'amortissement. Les algorithmes d'apprentissage par renforcement doivent équilibrer exploration, pour tester les différentes actions dans différents états afin de faire ressortir les paires état-action prometteuses, et exploitation, pour effectivement tirer parti des actions prometteuses en les choisissant souvent afin de faire émerger une politiques peaufinée ; cet équilibre peut par exemple être contrôlé par un facteur ϵ décroissant, avec l'agent d'apprentissage effectuant les actions avec la valeur action-état la plus élevée avec une probabilité ϵ de faire une action aléatoire. La taille des ensembles d'état et d'action pouvant être conséquentes (e.g. matrice de pixels RGB ou plateau de Go pour états, et combinaison de commandes de mouvement de robot ou pose de pierre sur un plateau pour actions), des réseaux de neurones profonds sont souvent utilisés comme approximateurs de fonction pour Q (DQN [47, 48]). D'autres algorithmes d'apprentissage existent en optimisant par exemple directement la politique (REINFORCE, PPO [49]) ou en découplant la politique et la fonc-

tion de valeur (*Actor-Critic* [50]). L'apprentissage par renforcement trouve des applications dans de nombreux domaines, tels que la robotique, les jeux, la planification de trajets, la gestion des ressources et la publicité en ligne.

Ces différents types d'apprentissage et d'applications sont supportés par différents types de modèles de réseaux de neurones profonds, tels que les réseaux neuronaux convolutifs [51] (CNN), indispensables en vision par ordinateur, les réseaux neuronaux récurrents [52] (RNN et LSTM) pour le traitement de données séquentielles, les réseaux antagonistes génératifs [53] (GAN, soit *Generative Adversarial Networks*) pour la génération de données réalistes, les réseaux neuronaux sur graphes [54, 55] (GNN), et les *transformers* [56] dont le mécanisme d'attention permet de donner une importance variable à différentes parties des données, améliorant ainsi la capacité du modèle à se concentrer sur les informations pertinentes.

2.2.3 Réseau de neurones sur graphes

L'idée fondamentale des réseaux de neurones sur graphes est de calculer une représentation vectorielle des noeuds d'un graphe en agrégeant les informations des noeuds voisins grâce à des opérations paramétrées, et donc propice à une tâche d'apprentissage. Soit h_v^0 le vecteur de *features* du noeud v . Des représentations vectorielles internes successives h_v^t sont calculées :

$$\forall t \in [0, N - 1], h_v^{t+1} = f_{\text{merge}_{\mathbf{W}_1^t}} \left(h_v^t, f_{\text{aggr}_{\mathbf{W}_2^t}}(\{h_u^t | u \in \mathcal{N}(v)\}) \right) \quad (2.4)$$

avec $\mathcal{N}(v)$ le voisinage de v , et f_{merge} et f_{aggr} des fonctions de fusionnement et d'agrégation paramétrées (par $\mathbf{W}_1^t$ et $\mathbf{W}_2^t$) et différentiables des vecteurs de features, par exemple concaténation ou somme pondérée. La définition du GNN peut également faire rentrer en compte des features sur les arcs. Les représentations vectorielles internes obtenues après N itérations peuvent être utilisées en entrée d'un réseau de neurones complètement connecté classique pour des tâches d'apprentissage au niveau des noeuds, des arcs ou du graphe entier.

Cette modélisation permet l'intégration de relations complexes au sein d'une tâche d'apprentissage grâce à la structure de graphe libre et riche en possibilités. Les représentations vectorielles sont obtenues par propagation d'information dans le graphe, ce qui permet aux GNN de capturer les interactions locales et globales entre les noeuds du graphe, et la structure exploitant uniquement les voisinages permet le traitement de graphes peu denses de grandes tailles. Le calcul de la représentation vectorielle des noeuds est également invariant aux permutations, ce qui est crucial pour les applications qui considèrent des entités qui n'ont pas d'ordre intrinsèque et permet ainsi un apprentissage plus adapté aux données non euclidiennes, au contraire d'images (matrices de pixels) ou de séries temporelles. Les opérations

définies pour les GNN sont également valides peu importe la taille du graphe ce qui permet de ne pas être restreint à toujours les mêmes structures de graphe, bien que la scalabilité aux graphes de différentes tailles soit toujours un défi pour la tâche apprise. Ces qualités font des réseaux de neurones sur graphes une structure adaptée pour le traitement de problèmes combinatoires définis ou représentables par des graphes en apprentissage automatique [57,58].

2.2.4 Mécanisme d'attention

Le mécanisme d'attention est un concept récent clé dans le domaine de l'apprentissage profond qui permet aux modèles de se concentrer sur des parties spécifiques des données d'entrée lors de la prise de décision. Inventé pour une meilleure précision dans des tâches de traduction [59], il est souvent utilisé dans les tâches de traitement du langage naturel et de la vision par ordinateur mais permet d'enrichir les capacités de beaucoup de modèles différents. Spécifiquement, l'architecture encodeur-décodeur avec mécanisme d'attention pour capturer simultanément les relations plus ou moins distantes dans une séquence (*transformers* [56]) plutôt que des réseaux de neurones récurrents permet un gain de performances considérable en traitement du langage naturel et la vision par ordinateur. Une adaptation des modèles de réseaux de neurones sur graphes intégrant le mécanisme d'attention est également possible en calculant en score d'attention entre les différents vecteur de features lors de la phase d'agrégation (GAT, *Graph Attention Networks* [60]) comme suit :

$$\forall t \in [0, N - 1], h_v^{t+1} = \sigma \left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} \mathbf{W}^t h_u^t \right) \quad (2.5)$$

Le score d'attention α_{vu} est donnée par

$$\alpha_{vu} = \text{softmax}_u(e_{vu}) = \frac{\exp(e_{vu})}{\sum_{k \in \mathcal{N}(v)} \exp(e_{vk})}$$

$$e_{vu} = \text{LeakyReLU}(\vec{a}^\top [\mathbf{W}h_v || \mathbf{W}h_u])$$

avec $\vec{a}$ et $\mathbf{W}$ des paramètres d'apprentissage.

2.3 Hybridation de l'apprentissage et de l'optimisation

Si la démocratisation de l'apprentissage profond est relativement récente, le recours à l'apprentissage en recherche opérationnelle n'est pas nouveau. En effet, des réseaux de neurones

étaient déjà utilisés à ces fins dans les années 1980 pour la génération de solutions pour le problème de voyageur de commerce par réseaux de Hopfield [61] ou réseaux de neurones autoadaptatifs [62, 63], ainsi que plusieurs autres applications pour la génération de solutions en optimisation combinatoire malgré les limitations techniques [64, 65]. Si la génération directe de solution est toujours un défi, des méthodes plus indirectes trouvent du succès, comme la découverte d’heuristiques gloutonnes performantes [66], la paramétrage automatique de solveur mathématique [67], ou le choix de noeud [68] ou de variables de branchement dans un algorithme de branch-and-bound [69, 70]. Le Chapitre 4 prolonge cette idée d’intégration dans l’exploration de l’arbre de branch-and-bound grâce à la génération de bornes d’optimisation performantes pour élaguer et combler l’écart d’optimalité. Les avancées précédemment citées tirent parti de l’apprentissage par renforcement pour la découverte automatique d’heuristiques ou l’exploration autonome de l’espace des possibilités [71], et des réseaux de neurones sur graphes pour traiter des problèmes paramétrés définis ou représentables par des graphes [57]. Un autre paradigme prometteur est également la source de progrès récents : plutôt que de prédire en imitant un expert et donc en minimisant une erreur de prédiction, la structure d’un problème d’optimisation peut être utilisée pour concevoir de meilleurs modèles de prédiction en exploitant l’erreur de décision [72, 73]. Dans cette optique, Chapitre 7 cherche à générer des valeurs en entraînant directement à maximiser la borne duale obtenue et non pas en imitant des valeurs connues. D’un point de vue plus pragmatique, le passage de la théorie à la pratique et l’application directe de l’apprentissage machine en industrie n’est pas une chose aisée tant à cause de la qualité des données, la complexité des modèles ou la robustesse des résultats liés à l’apprentissage [74], et nous étudions Chapitres 5 et 6 des applications pratiques, le repositionnement d’ambulances et l’insertion de rendez-vous dans des tournées de véhicules optimisées, en relation avec des utilisateurs finaux non formés à l’apprentissage machine ni l’optimisation.

CHAPITRE 3 DÉMARCHE ET ORGANISATION DE LA THÈSE

Cette thèse est organisée autour de 2 articles publiés et 2 chapitres, tous faisant partie d’une démarche de recherche et réflexion commune concernant les applications de l’apprentissage automatique à des problèmes classiques en optimisation mathématique, théoriques ou concrets. Les articles et projets résultent de recherche universitaire et de collaborations industrielles lors de stages de recherche Mitacs chez ElementAI (maintenant ServiceNow) et Fastercom à Montréal.

Le Chapitre 4 présente le premier article, intitulé *Improving Branch-and-Bound using Decision Diagrams and Reinforcement Learning* et publié dans *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (CPAIOR). Celui-ci est la continuation du travail de Professeur Quentin Cappart, alors post-doctorant, et de l’article *Improving optimization bounds using machine learning : Decision diagrams meet deep reinforcement learning* publié dans *AAAI Conference on Artificial Intelligence* qui présente les diagrammes de décision comme méthode de génération de bornes pour les problèmes d’optimisation discrète, plus flexible par exemple que la relaxation linéaire classique. Cette flexibilité et la nature séquentielle de la construction d’un diagramme de décision sont exploités par une approche d’apprentissage par renforcement pour resserrer les bornes générées. Ces bornes sont notamment utiles au sein d’un algorithme de branch-and-bound qui va permettre de les exploiter pour élaguer des parties de l’arbre de recherche, permettant ainsi une recherche plus efficace et une résolution plus rapide des problèmes combinatoires considérés. L’article présenté est le résultat de l’intégration des bornes d’apprentissage dans un algorithme de branch-and-bound basé sur les diagrammes de décisions, mettant une évidence une réduction du nombre de noeuds explorés allant jusqu’à 66% par rapport au cas d’utilisation d’une heuristique classique jugée performante. Cette réduction est néanmoins contrebalancée par le temps d’exécution total très élevé, dû au coût des appels répétés à l’agent entraîné. Une application plus aboutie et techniquement performante, ainsi que la publication d’un code open-source permettant des résolutions jusqu’à 15% plus rapides pour le problème du stable maximum, sont présentés dans la partie *Application to a Branch-and-Bound Algorithm* du papier *Improving Variable Orderings of Approximate Decision Diagrams using Reinforcement Learning* publié dans *INFORMS Journal on Computing* et présenté Annexe A.

Le Chapitre 5 correspond au deuxième article, intitulé *Repositioning Fleet Vehicles : a Learning Pipeline* et publié dans *International Conference on Learning and Intelligent Optimiza-*

tion. Il est le résultat d’une collaboration avec l’entreprise montréalaise Fastercom spécialisée dans l’optimisation de tournées, les logiciels de gestion pour flottes de véhicules et la prise de rendez-vous automatisée. La gestion de flotte sous incertitude n’est pas une tâche aisée et repose en pratique sur les compétences et l’expérience de répartiteurs en contact direct avec les véhicules d’une flotte. Nous développons une solution permettant d’assister ces répartiteurs grâce à une approche d’apprentissage simple, aisément explicable et interprétable, qui tire parti de données passées pour estimer le besoin en véhicules dans différentes parties d’une zone d’opération en fonction de l’heure de la journée et de l’activité récente de la flotte. Cette application est spécifiquement destinée à une flotte d’ambulances, et la prise en compte de motifs médicaux est donc intégrée puisque ceux-ci ont un impact sur les requêtes futures. Des performances de l’ordre de 10 à 20% en réactivité sont constatées, mais celle-ci sont à contrebalancer avec la distance totale parcourue qui peut augmenter jusqu’à 10% en fonction des objectifs des répartiteurs. Un déploiement des algorithmes développés a eu lieu avec une entreprise privée en Belgique et d’autres déploiements sont à l’étude. Le succès de ce projet et de la publication associée a mené au développement d’un nouveau projet avec Fastercom, présenté Chapitre 6.

Le Chapitre 6 présente un projet visant à accélérer la suggestion de créneaux de rendez-vous en temps réel chez des partenaires industriels. Une prise de rendez-vous, avec par exemple des infirmières à domicile ou des techniciens en bâtiment, dépend d’un programme d’optimisation de planification et de tournée de véhicules. Afin d’éviter des résolutions successives coûteuses pour proposer des créneaux de rendez-vous à un client et afin de pouvoir répondre en temps réel à partir d’une journée déjà optimisée, nous proposons d’étudier le développement d’un prédicteur quantifiant la qualité d’une insertion de client dans un problème de tournée de véhicule (VRP, *vehicle routing problem*) pour directement proposer ces prédictions ou pour limiter l’usage de l’optimisation avant de revenir au client. Nous développons une représentation d’un problème de VRP et d’une solution sous forme de graphe afin d’en avoir une représentation riche et exploitable par un réseau de neurones sur graphes. L’utilisation de données historiques et de résultats d’optimisation permet de mettre en place un apprentissage adapté à un prestataire et un solveur de VRP. Nous montrons qu’un prédicteur entraîné permet effectivement d’identifier les créneaux prometteurs et d’éviter systématiquement les insertions irréalisables, permettant ainsi d’envisager un déploiement opérationnel de l’outil.

Finalement, le Chapitre 7 présente un projet d’apprentissage non-supervisé pour la génération de facteurs lagrangiens au sein de la relaxation de Held et Karp pour le voyageur de commerce. L’utilisation d’apprentissage non-supervisé cherchant à augmenter la valeur des bornes générées permet de guider la génération des facteurs lagrangiens sans connaître a priori quels seraient de bons facteurs. L’intégration du générateur de borne duale entraîné au

sein d'un algorithme de branch-and-bound permet de démontrer l'intérêt de l'apprentissage pour la résolution de problèmes combinatoires, avec un gain temporel sur une approche de résolution classique. Une publication est prévue durant l'été 2023.

CHAPITRE 4 ARTICLE 1 : IMPROVING BRANCH-AND-BOUND USING DECISION DIAGRAMS AND REINFORCEMENT LEARNING

Augustin Parjadis^{ab}, Quentin Cappart^{ab}, Louis-Martin Rousseau^{ab}, David Bergman^c

^aÉcole Polytechnique de Montréal, Montreal, Canada, ^bCIRRELT, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, ^cUniversity of Connecticut

Cet article a été soumis à *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research* le 28 janvier 2021 et a été publié le 17 juin 2021.

ABSTRACT

Combinatorial optimization has found applications in numerous fields, from transportation to scheduling and planning. The goal is to find an optimal solution among a finite set of possibilities. Most exact approaches use relaxations to derive bounds on the objective function, which are embedded within a branch-and-bound algorithm. Decision diagrams provide a new approach for obtaining bounds that, in some cases, can be significantly better than those obtained with a standard linear programming relaxation. However, it is known that the quality of the bounds achieved through this bounding method depends on the ordering of variables considered for building the diagram. Recently, a deep reinforcement learning approach was proposed to compute a high-quality variable ordering. The bounds obtained exhibited improvements, but the mechanism proposed was not embedded in a branch-and-bound solver. This paper proposes to integrate learned optimization bounds inside a branch-and-bound solver, through the combination of reinforcement learning and decision diagrams. The results obtained show that the bounds can reduce the tree search size by a factor of at least three on the maximum independent set problem.

4.1 Introduction

Historically introduced for encoding Boolean functions and used for circuit design and verification [20, 21], *Decision Diagrams* (DDs) have recently been reapplied in the field of combinatorial optimization [23, 75, 76], for example to sequencing problems [24] or the multidimensional bin packing problem [25]. Assuming a maximization objective, the optimal solution can be obtained in polynomial time in respect to the size of the decision diagram by following

the longest path from the the root to the terminal node of an exact DD. However, the size of exact DDs grows exponentially with the number of variables, which make them unsuitable for solving large problems. Recently, decision diagrams provided new means of obtaining bounds for combinatorial optimization problems that can be significantly better than those obtained via a traditional linear programming relaxation [77]. Bergman et al. [78] proposed the use DDs to encode a parametrizable and tractable approximation of the solution set. Such structures are referred to as *approximate* DDs.

The performance of this procedure highly depends on the ordering of the variables used to create the DDs. Better ordering can lead to tighter optimization bounds, which results in fewer nodes explored during the BnB search and an expected solution time reduction. Nevertheless, finding an ordering that yields the best bound is NP-hard and difficult to model. Typically heuristics have been considered for defining variable ordering [29] and only a few limited studies have proposed exact approaches for specific problem classes (see, e.g. [33]). Recently, Cappart et al. [28] proposed a deep reinforcement learning (DRL) [79, 80] approach for computing the variable ordering. The idea is to train an agent to build a DD, with the incentive (i.e., the reward) to obtain bounds as tight as possible. However, this procedure has not been integrated in a BnB algorithm.

The contribution this paper makes is to illustrate the benefits and challenges of using both primal and dual bounds obtained with DRL-guided DDs within a BnB procedure. As a first experiment we apply this algorithm to the *maximum independent set problem* (MISP). Our preliminary results show that the proposed approach is able to prove optimality with significantly fewer nodes explored, due to the better bounds obtained. However, it is done at the expense of a significant increase in the execution time, as a deep neural network has to be called multiple times at each BnB node. An improved algorithm which uses caching and restricts the use of the DRL agent is also presented, and proves efficient on larger problems. This paper introduces some preliminaries for DDs and reinforcement learning before presenting the BnB algorithm, followed by experimental results. Finally, the limitations of the current approach and directions for future research are discussed.

4.2 Learning Bounds Inside Branch-and-Bound

4.2.1 Decision Diagram-Based Branch-and-Bound

The DDs used in this work are Binary Decision Diagrams (BDDs), although the generalization to multi-valued decision diagrams is immediate. A BDD $B = (U, A, d)$ is a directed acyclic multi-graph in which the node set U is partitioned into layers $L_1, \dots, L_{n+1}$. The set

of solutions for a problem can be represented by a BDD in which each path from the root node r to the terminal node t encodes a feasible solution : each arc along the path gives the value of the variable associated with the layer the arc starts from, and a longest path in the diagram gives an optimal solution to the problem. Conversely, each feasible solution can be found in the BDD. Each layer $L_1, \dots, L_n$ is associated with a unique variable x_k of the problem with $k \in \{1, \dots, n\}$. Each arc $a \in A$ goes from one layer to the next and has label $d(a) \in \{0, 1\}$ that encodes the values of the layer's associated binary variable.

For larger problems in combinatorial optimization, BnB algorithms are widely used to generate a search tree of manageable size using optimization bounds. Such bounds can be obtained via feasible solutions and linear relaxation, and we focus here on how approximated DDs provide a simple alternative for computing bounds. An exact DD can be *relaxed* by merging nodes of a layer to narrow the diagram without removing any solutions, but adding unfeasible solutions [78]. A relaxed DD provides a dual bound when solved, the quality of which depends on the amount of merging operations done. Likewise, a *restricted* DD can be created by removing nodes of an exact DD ; some solutions are deleted but none are created, yielding a subset of solutions that can provide a primal bound. A branching process can also be conducted on DD nodes, which provides a complete BnB scheme based on DDs as an alternative to the classic linear programming-based BnB. A DD-based BnB algorithm is described in Algorithm 2. Detailed information is proposed by Bergman et al. [29].

4.2.2 Variable Ordering and Reinforcement Learning

Recently, a machine learning approach has been proposed to address the NP-hard problem of variable ordering for DDs [28] using reinforcement learning (RL) [81]. The goal of a RL agent is to maximize the expected sum of rewards obtained by learning a behavior policy. In the case of DD construction, the agent incrementally builds an approximated DD and observes the bound given by the partially built diagram. The rewards given as feedback are defined by the evolution of the bound obtained (a reduction of an upper bound is encouraged, whereas an increase is penalized and vice versa for a lower bound). The state observed by the agent is a function of the problem instances and of the variables already added in the current DD. An action consists in selecting a new variable to add to the next layer of the DD. Finding a policy maximizing the action-value function for all states and actions is hard and a practical solution is to compute an approximation of Q using a Q -learning algorithm [46] applied to a graph neural network. The exact procedure can be found in [28].

4.2.3 The Branch-and-Bound Algorithm with a RL Agent

This section presents the main contribution of the paper : how to implement a BnB algorithm using DDs together with RL. The process is as follows. At each node explored during a DD-based BnB algorithm, a relaxed and a restricted DD are built and an ordering for the variables left at this stage of the search has to be determined for each DD. Currently, heuristics that try to greedily limit the width of the DD are used, and perform much better than simple lexicographic or random orderings.

We propose to work on the ordering of the approximate DDs by using a RL agent on a graph embedding rather than handcrafted heuristics. Two steps are to be considered for building this agent : (1) a training phase, consisting of learning a good policy for the problem at hand, and (2) a solving phase, corresponding to the execution of the BnB algorithm.

Vectorized representation

A graph neural network (GNN) [55, 57] is used to obtain a vectorized representation of a graph by computing node embeddings. The vectorization is built as follows : (1) the problem instance is represented as a graph (e.g., a MISP instance), (2) each vertex/edge of the nodes are decorated with relevant features (e.g. the weight of each vertex), (3) A GNN is used to obtain an d -dimensional embedding for each vertex of the graph, and (4) the embedding is given as input to a fully-connected neural network in order to obtain the final Q-values that are used for the prediction, indicating the quality of each vertex to be inserted in the current embedding.

Training Phase

The training is based on neural fitted Q -learning with a set of randomly generated graphs, and returns the weights $\mathbf{w}$ for the approximated action-value function $\hat{Q}$. The Q -value approximation is obtained with the use of a GNN with the library `structureToVec` [82]. The standard Q -learning algorithm can be enriched in several classic ways, among which are mini-batches to guarantee a better gradient descent based on several examples instead of one, reward scaling for a better handling of large reward quantities and an adaptive ϵ -greedy policy to move away from a local optimum, balancing exploration and exploitation. Details on the learning phase can be found in [28]. As the training uses the bound obtained via the partially built DD, the RL agent is dependent on the type of bound used. Therefore we consider two agents, one trained for relaxed DDs for which a low value upper bound is desired, and the other for restricted DDs for which the highest possible lower bound is desired.

Once the parameters $\mathbf{w}$ for the Q -value have been learned, a simple policy can be derived as described below.

Solving Phase

The solver uses a trained RL agent each time a DD is developed. Algorithm 1 describes how an ordering is obtained by calling the agent several times, and Algorithm 2 describes the BnB algorithm using the DDs constructed with the RL agents. New variable orderings are computed at each node of the BnB to build the restricted and relaxed DDs. The RL agent is designed to obtain high-quality bounds, which then allows to update the lower bound or prune the node if applicable. Let B_{ut} be the DD induced by all nodes and paths going from u to t (which gives in particular $B_{rt} = B$). If B_{ut} is an exact DD, B'_{ut} designates a restriction and $\bar{B}_{ut}$ a relaxation of B_{ut} . If a restricted DD B'_{ut} is exact, no restriction was needed and thus no branching is performed. The variable orderings are obtained as described below.

Algorithm 1 DD construction with RL agent

```

1:  $B = (\{L_1, \dots, L_n\}, A, d)$  an empty DD, to be built with the variables  $V$ 
2: for  $j \in \{1, n\}$  do
3:    $u = \operatorname{argmax}_{a \in V} \hat{Q}(s, a, \mathbf{w})$ 
4:   build layer  $L_j$  with the variable  $u$ 
5:    $V \leftarrow V \setminus \{u\}, b_u = 1$ 
6: end for
7: return  $B$ 

```

In Algorithm 1, the state of the problem is given to the agent that responds with the Q -values for each variable. The variable u that maximizes the Q -value learned by the GNN is chosen and added to the partially built DD. A feature b_u for each $u \in V$ indicates if a variable has already been selected for the DD construction. After the selection of the variable u , the feature is updated. This modifies the state of the problem, so the Q -values are computed again to select the next variable. The process continues until all the variables have been considered and the DD is complete.

Complexity

Obtaining a variable ordering by calling a GNN is expensive because of the nature of the steps involved. The node embeddings are created by a message passing algorithm and fed to a fully connected neural network. This has to be done for each variable in the ordering, resulting in a time complexity of $\mathcal{O}(|E| \times n^2)$. In the next section, Algorithm 1 for variable ordering is compared against a classic heuristic called *MinState*, which has a time complexity

Algorithm 2 DD based branch-and-bound, 2 RL agents

```

1:  $z_{\text{opt}} = -\infty$ ,  $\mathcal{Q} = \{r\}$ 
2: while  $\mathcal{Q} \neq \emptyset$  do
3:   select node  $u \in \mathcal{Q}$ 
4:   create restricted DD  $B'_{ut}$  with Algorithm 1
5:   if  $v^*(B'_{ut}) > z_{\text{opt}}$  then  $z_{\text{opt}} = v^*(B'_{ut})$  end if
6:   if  $B'_{ut}$  non exact then
7:     create relaxed DD  $\overline{B}_{ut}$  with Algorithm 1
8:     if not pruned, select exact cutset to add to  $\mathcal{Q}$ 
9:   end if
10: end while

```

of $\mathcal{O}(w \times n)$; with w the width of the DD. The additional complexity brought by the GNN should thus have to be justified with an efficient node reduction during the BnB.

4.3 Experimental Results

The goal of the first set of experiments is to show that a significant reduction of nodes explored in the tree search is possible with a well trained GNN. However, it is done at the expense of the computation time. The second experiment shows how this issue can be mitigated using a caching mechanism, and a hybrid heuristic. Similar to previous works [28, 83], the case study considered is the maximum independent set problem (MISP). The solver is written in C++ and is implemented upon the code of Cappart et al. [28] for the RL agent part and Bergman et al. [29] for the DD-based BnB. The learning and the solving were ran on a Dell XPS 15 9570 with a Intel i7-8750H 2.20GHz CPU. The training time was limited to eight hours. All instances considered have been randomly generated using a Barabasi-Albert scheme with the density parameter $\nu = 4$ [84].

Definition (Maximum Independent Set Problem). *An independent set I in a weighted graph $G = (V, E)$ is a subset $I \subseteq V$ in which 2 vertices cannot be adjacent. The problem consists of finding an independent set of maximum weight, i.e. maximizing the function $f(x) = \sum_{k=1}^{|V|} w_k x_k$ such that $\forall (i, j) \in E, x_i + x_j \leq 1$ with x_k indicating whether the vertex k is selected or not, and w_k being the weight associated to this vertex.*

4.3.1 Performances of the Learned Variable Ordering

We plot in Figure 4.1 the number of nodes explored during the DD-based BnB over 100 random instances. For the variable orderings, the GNN algorithm uses a GNN trained for relaxation over random graphs with the same distribution. Restriction is still done with

a MinState ordering. The GNN consists of 4 iterations of belief propagation and a neural network with 2 layers of size 64. The *heuristic* algorithm uses MinState for relaxation and restriction. The results can be found in Table 4.1, with the average time in seconds. We can see that even if the number of nodes is drastically reduced, the execution time remains an important concern compared to MinState. Figure 4.1 shows the percentage of the graphs solved under a given number of nodes in performance profile [85] : the higher the curve, the better. Overall, the number of nodes needed to solve all the MISP instances is typically reduced by a factor of three to four with the RL agent for graphs over 100 nodes, noting that 90% of the instances were solved in fewer nodes.

TABLEAU 4.1 Average time and nodes explored for 100 MISP instances of size n

Algorithm	$n = 80$		$n = 100$		$n = 120$	
	Nodes	Time	Nodes	Time	Nodes	Time
GNN	1416	16.70	4628	79.90	34434	878.10
GNN+	2131	2.40	13152	10.00	81015	41.10
MinState	2802	0.32	22680	5.30	100822	40.40

4.3.2 Caching to save computation time

To avoid calling the agent too often and speed-up the resolution, the GNN-computed orderings are stored and re-used for subsequent DDs that might have a similar structure. To re-use an ordering, the number of variables for the current DD has to be close to the number of variables of the stored ordering and be included in it. When the number of variable differs too much, the stored ordering can be quite poor for the current DD (similar to a random ordering). This sacrifices some precision in favor of computational efficiency. Furthermore, the computation time bottleneck is expressed the most when it is needed the least, which is at the end of the BnB tree for small DDs. Bound improvements there are small and do not have major consequences, and small DDs are very fast to build with little need for a GNN. We experiment on the use of the RL agent before a given threshold for the number of nodes (here, 100 BnB nodes), and the use of the MinState heuristic after this threshold to close the search faster. Those improvements are natural given the cost of a GNN feed-forward operation, and we observe in Table 4.1 with the algorithm GNN+ that the average number of nodes explored during a search can in fact be effectively reduced while keeping a competitive solving time for large problems.

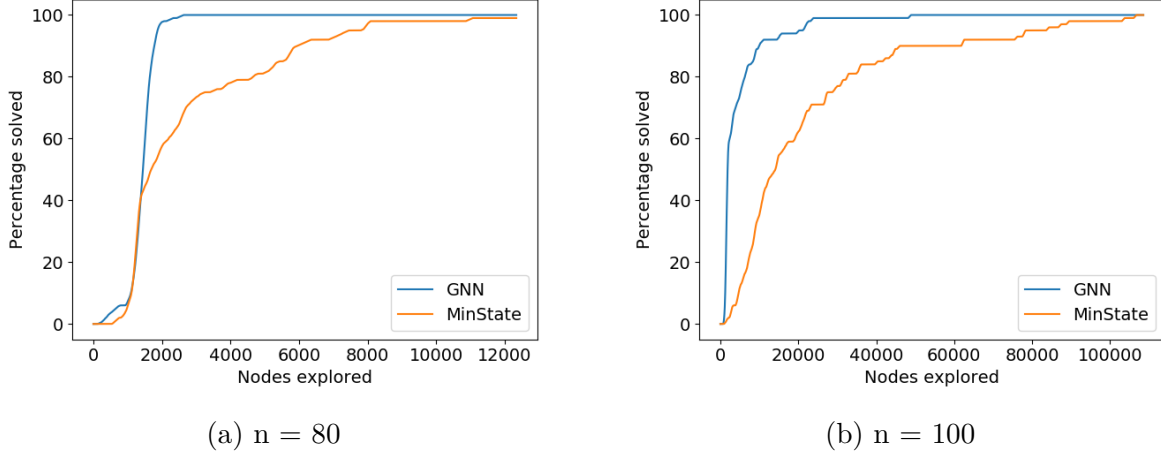


FIGURE 4.1 Performance profile of a GNN agent and MinState for variable ordering over 100 instances of the MISP

4.3.3 Discussion

As highlighted in the experiment, the learned bounds can be successfully leveraged in the solver in order to reduce the number of nodes explored. However, the execution time is an important bottleneck, which makes the hybridization challenging to design, and additional mechanisms should be considered in order to improve the efficiency. *Caching* is one of them but is still not enough. We made other attempts to get improvements, such as (1) using the same agent for the restriction and the relaxation, (2) using a fixed ordering during the complete BnB process instead of dynamically recomputing it, (3) calling the GNN only every n steps or at random intervals, but without much success. We believe that finding other ways to improve is a promising research direction. A similar question has been addressed by Gupta et al. [86] for standard MIP solvers. They show that expressive and costly GNNs can be combined with inexpressive but cheap fully connected neural networks in order to obtain a better trade-off between prediction quality and computational efficiency.

4.4 Conclusion

DDs provide a new flexible and general methodology for generating tight bounds for optimization problems, and DD-based BnB algorithms are often competitive with recent integer programming solvers. This paper presents a method that takes advantages of the flexibility of DDs for bound generation in BnB through machine learning. The reduction of the number of nodes indicates that this is a promising research direction, but subject to important challenges (e.g. the execution time). The generic nature of DD construction and BnB sol-

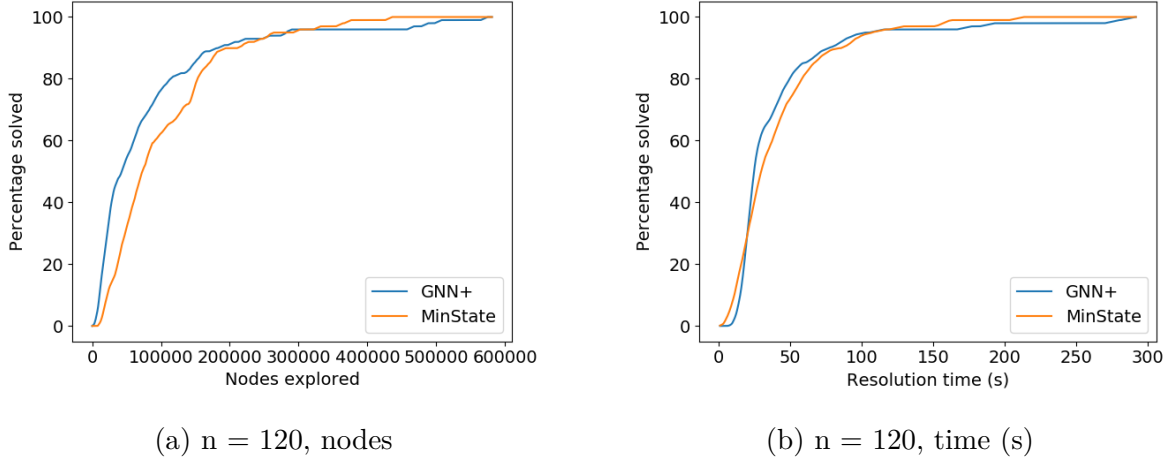


FIGURE 4.2 Performance profile of an improved GNN agent and MinState for variable ordering over 100 instances of the MISP

ving for discrete optimization problems makes for an interesting combination, with possible application to other problem classes like the maximum cut problem [77], the set covering problem [78] and the traveling salesperson problem [87, 88].

The next step to better bridge the gap between machine learning and optimization would involve a general framework that takes advantage of the graph structure and recursive formulation of a problem by learning to construct DDs with high quality bounds for the problem, and solving it with an efficient BnB algorithm. Data generation for the training step would remain a challenge.

The use of machine learning to generate both lower and upper bounds for combinatorial optimization proved efficient and opens the door to multiple possibilities for future research on the integration of these disciplines. Applying deep learning tools to optimization faces practical bottlenecks that have to be overcome for practical applications and broad adoption in exact optimization solvers.

CHAPITRE 5 ARTICLE 2 : REPOSITIONING FLEET VEHICLES : A LEARNING PIPELINE

Augustin Parjadis^{ab}, Quentin Cappart^{ab}, Quentin Massoteau^c, Louis-Martin Rousseau^{ab}

^aÉcole Polytechnique de Montréal, Montreal, Canada, ^bCIRRELT, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, ^cFastercom, Montreal

Cet article a été soumis à *International Conference on Learning and Intelligent Optimization* le 1 février 2023 et a été accepté pour publication le 15 mars 2023.

ABSTRACT

Managing a fleet of vehicles under uncertainty requires careful planning and adaptability. We consider a *ride-hailing problem* where the operator manages vehicle repositioning to maximize responsiveness. This paper introduces a supervised learning pipeline that uses past trip data to reposition vehicles while adapting to fleet activity, a geographical zone, and seasonal or daily request variation. The pipeline incorporates trip features, such as medical motives of transportation for ambulances and the time and location of the trips. This provides a better estimate of the probability that a given vehicle will be required in a particular sector and provides insights into which events and trip features should be incorporated into the decision-making process for better fleet management and improved reactivity. This tool has been developed for, and used by, operators of an ambulance company in Belgium. Using predictors for ambulance repositioning reduces at least 10% of the overall fleet reaction distance.

5.1 Introduction

Fleet management in a ride-hailing context is a challenging problem due to the stochastic nature of the demand and the dynamic decisions that must be made. Medical transportation for unplanned requests faces the same challenges and aims at providing a wide-spread and fast coverage to ensure medical services availability ; they have been a regular focus of routing problems in general, as cities need efficient and cost-effective medical transportation [89, 90].

This work focuses on learning from limited historical data for ride-hailing in a medical transportation setting and providing easy tools to set up, interpret and apply to small fleets of ambulances. The contribution of this paper is an effective learning pipeline that can provide an adaptable repositioning agent capable of recognizing daily and hourly patterns and adap-

ting to evolving demand distribution. The current strategy used by the company taking care of the repositioning yields a low total distance driven by the ambulance fleet but provides poor reactivity to requests ; our learning pipeline improves this without a significant increase in the total distance driven on real-world data. The paper is organized as follows. Section 5.2 reviews the relevant literature. Then, Section 5.3 introduces and formalizes the ride-hailing problem. Section 5.4 presents the pipeline structure. Section 5.5 presents a case study where we apply our repositioning agent to a fleet of ambulances in Belgium for unplanned medical trips.

5.2 Literature review

Vehicle routing problems (VRPs), focusing on finding optimal routes for vehicles between multiple locations and under various constraints, are ubiquitous in the flow of people and goods in our increasingly interconnected cities, networks, etc. As a traveling salesman problem at its core, these problems are hard [91,92]. They are often subject to additional constraints when modeling real problems, often linked to time windows, traffic rules, environmental considerations, or many other potential obstacles [93–95].

Ride-hailing problems are vehicle routing problems, more precisely dial-a-ride problems where multiple users request trips with given time windows and are transforming logistics and sustainability in the context of rising awareness towards efficiency and environmental issues. This transformation of urban mobility can stimulate the adoption of new technologies and positively impact energy consumption and quality of life [96,97]. Nevertheless, the success of these systems depends on their effective efficiency and ease of use, which is a challenge given the complex nature of vehicle routing problems in dynamic environments. Extensive literature can be found for dynamic VRPs and dial-a-ride problems in recent surveys [98,99]. The dynamic nature of ride-hailing problems, with new requests arriving at unknown times and locations, creates different challenges and requires novel solution approaches than do not simply assume perfect information.

The first challenge of ride-hailing problems is to assign vehicles to incoming requests. This has been tackled by traffic flow study [100], oversupply analysis [101], and negotiation techniques between vehicles [102]. Several studies approach fleet management from the opposite direction, focusing on vehicle repositioning to anticipate future requests. Miao et al. [103] use a receding horizon for distribution estimation, Zhang and Pavone [104] focus on a queuing-theoretical model for supply rebalancing, while Braverman et al. [105] introduce fluid-based optimization methods to control car flow and maximize the expected value of served requests. Other works study assignment and repositioning for a more complete approach to the ride-

hailing problem with either heuristic strategies at global scales for autonomous vehicles [106] or large-scale optimization-based methods [107–109]. Furthermore, increasing interconnectivity and data availability allow enriched models and more precise control of the vehicles. For example, real-time information such as traffic congestion or driving times opens up the possibility of adapting programmed tasks and trip trajectories [110, 111].

Recent machine learning developments allow the creation of new tools to tackle complex problems and integrate historical data. From the plethora of regressors and classifiers, [112], multi-layer perceptrons and deep learning emerged as powerful tools to learn from unstructured data [79]; deep reinforcement learning (DRL) [80, 81] allowed the training of agents to interact with their environment. These approaches are incredibly prolific in image classification [113], natural language processing [114], super-human performance in games [115], and have also been considered by the operations research community for mathematical optimization [116]. Optimization coefficients can, for example, be learned inside a more traditional learning algorithm [117], multi-agent DRL allows the training of a fleet of agents to maximize shared profit [118, 119], and can integrate traditionally complex variants like electric vehicle charging [120]. The ride-hailing problem can thus be tackled with a hybrid approach of learning, and optimization [121], or learning from optimization models [122]. Handling a whole fleet in a dynamic environment is a complex task, often solved by reinforcement learning in which fleets are represented by an agent [123], or multiple agents [124, 125], that explore and refine fleet management policies. These learning approaches are efficient but require a large amount of data and are not easy to use and interpret for dispatchers of a small fleet. Based on this context, our contribution focuses on a smaller granularity, both time-wise and vehicle-wise, aiming at developing a pipeline that works with a limited amount of data and is easy to interpret for the operators.

5.3 The Ride-hailing Problem

This section presents the ride-hailing problem and the relevant notation used throughout this work.

5.3.1 Problem Definition

Let V be a fleet of vehicles and R the set of unplanned requests arriving throughout the day. A vehicle $v \in V$ is characterized by its geographical position and its status. The possible status are as follows : (1) driving to a client, (2) serving a client, (3) repositioning, (4) idling at a location, and (5) waiting for repositioning instructions. A request $r \in R$ is emitted at a

time r_t and is characterized by its origin and destination coordinates.

Each time a request arises, it is assigned to the closest vehicle. Vehicles idling or repositioning immediately go serving the request they have been assigned. However, vehicles currently serving a request might also be chosen to serve the new request after their current ones have been fulfilled, as there is no maximum waiting time. The requests can arise and lead anywhere, usually hospitals and patient houses; they will all get served by the fleet.

A repositioning decision is made when a vehicle finishes serving a request and does not have another to serve next. Let L be the set of locations containing predefined geographical points where vehicles are sent to idle between requests. The requests can arise anywhere, whereas vehicles must wait at the predefined locations L . The goal is to reposition vehicles to the idling locations to minimize the average reaction distance, defined as the distance driven by an ambulance from where it is when assigned to a request to the pick-up point of the request. The reaction distance includes the distance driven by ambulances that have to finish current trips before answering the request.

Visual Example

An illustration of possible decisions is proposed in Fig. 5.1 : (1) A request for a patient is first received and assigned to a nearby vehicle chosen during the assignment process. (2) After the ride, the vehicle is repositioned to a location chosen during the repositioning process. (3) Let us point out that the location does not have to be the closest one, nor does it have to be its original idling location. (4) Finally, another vehicle can be repositioned to balance the supply.

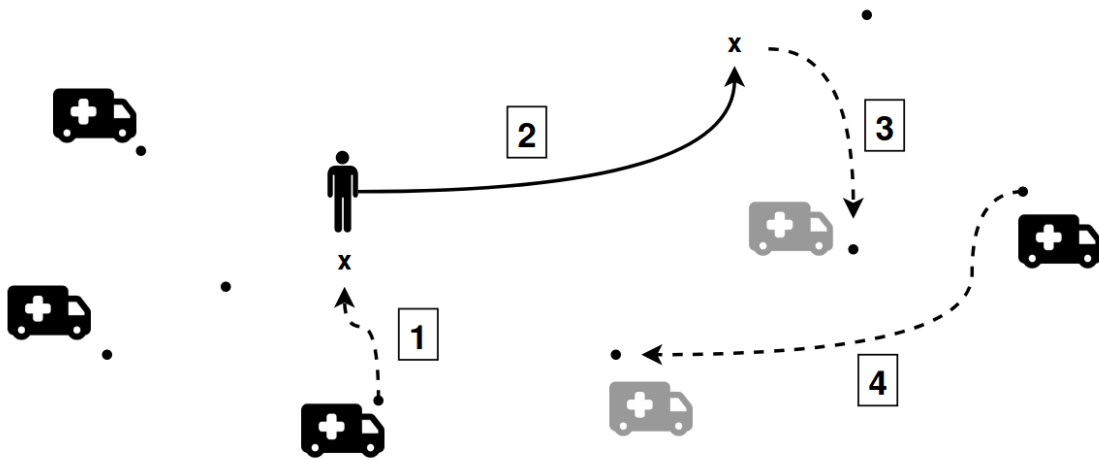


FIGURE 5.1 Ride-hailing situation - example.

5.3.2 Modelling the Repositioning Task

A *decision* must be carried out each time a new request is received and each time a vehicle finishes serving a trip request. An *episode* refers to a full day of operations of the fleet and contains as many decisions as requests received and trips finished. When a decision must be made, the system is given a *state*. The formal definitions of a state and a decision are proposed below.

States

A state $\mathbf{s} = (s_R, s_r, s_t, s_V)$ is composed of a prior requests vector (s_R), a request vector (s_r), a time vector (s_t) and a vehicle vector (s_V). The prior requests vector is composed of tuples indicating the location of origin, arrival, and the medical motive of the past requests within a given horizon $H : (r_i^O, r_i^A, r_i^m)_{i \in H}$. The request vector s_r provides the geographical coordinates of the origin and destination of the request received at the start of the decision epoch. It is empty if the epoch was triggered by a vehicle becoming idle. The time vector $s_t = (d, t)$ indicates the day of the week d and hour of the day t as one-hot encoded vectors. The vehicle vector s_V indicates the status and positions of the fleet's vehicles.

Decisions

A repositioning decision $a = (a_1, a_2, \dots, a_V)$ is composed of vehicle repositioning instructions. For all vehicles, $v \in V$, the instruction $a_v \in L \cup \{\emptyset\}$ indicates the location $l \in L$ at which a vehicle must reposition or an absence of instruction. Idle vehicles that finished serving a trip request must be repositioned to idle at a valid location.

5.4 Learning Pipeline for Vehicle Repositioning

This section presents a learning pipeline used in the specific context of ambulance management, which is general enough to be adapted to other situations. The pipeline consists of a learning algorithm and a prediction agent. It is illustrated in Fig. 5.2 and is divided into 4 steps : (1) Past data is collected, used to build context vectors, and split to build the training and test sets, (2) an agent is trained on this dataset and is used operationally to predict the next location most likely to receive a request, and (3) this information is fed to a repositioning algorithm adapted to the fleet. Each step is detailed in the next paragraphs.

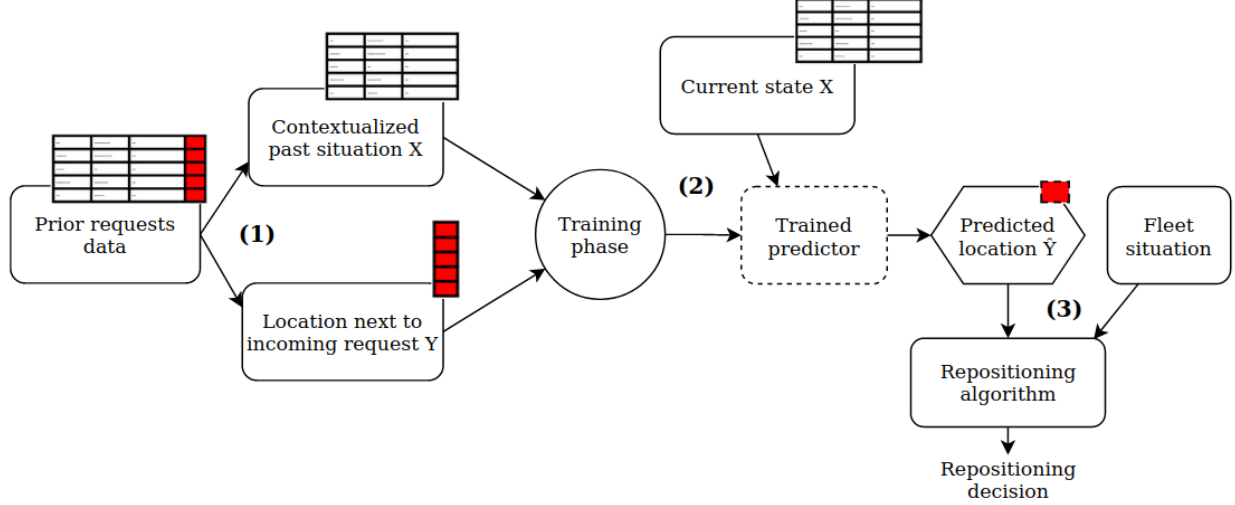


FIGURE 5.2 Pipeline structure.

Step 1 : Data Preparation

The data comprises the set of feature vectors X , encoding the state and context in which a trip request arrived, and the associated set of target values vector Y , containing the locations of the origin of the requests. Several features are used to make a prediction, such as time, location, and motives of previous requests. Geographical distributions will, for example, be affected by the population moving during work days or days off, resulting in hourly and weekly variations in requests. Previous requests' origin, destination, and motives are also likely to provide relevant information about whether a patient will be emitting another request and when; we encode their frequency and recency with discounted factors in a context vector that aggregates the information from H requests as follows. Prior requests origins (PRO), prior requests arrivals (PRA), and medical motives are features computed from the components of the prior requests vector. For each location l , we define the prior requests origins feature $s_l^O = \frac{1}{H} \sum_{i \in [1, H]} \delta(l, r_i) \beta^i$, where $\beta \in]0, 1]$ is a constant, and

$$\delta(l, r_i) = \begin{cases} 1 & \text{if } l = r_i^O \\ 0 & \text{otherwise} \end{cases} \quad (5.1)$$

indicates which location a prior request emerged next to, with r_i^O being the closest location from r_i 's origin. Here, r_1 is the latest request at time t , r_2 the one before it, etc., with r_H being the oldest request considered on our horizon. For the requests arrivals, $(s_l^A)_{l \in L}$ is computed similarly with arrival locations. For each medical motive $m \in M$, we define

$s_m = \frac{1}{H} \sum_{i \in [1, H]} \mu(m, r_i) \beta^i$, with

$$\mu(m, r_i) = \begin{cases} 1 & \text{if } m = r_i^m \\ 0 & \text{otherwise} \end{cases} \quad (5.2)$$

where r_i^m is the motive of request i , indicating the number of requests with motive $m \in M$ and their recency. $(s_l^O)_{l \in L}$, $(s_l^A)_{l \in L}$, and $(s_m)_{m \in M}$ thus indicates which hospitals requests emerged from and arrived at, and which medical motives were observed. The set of medical motives M can include simple consultations, transfers, specialized interventions, etc. Table 5.1 provides a summary of the state features for our case study related to healthcare. All feature values are in $[0, 1]$. The relevance of such features is analyzed in Appendix B.

The training set is built by taking whole days of data and extracting the arising requests as well as the state in which they arrived to build the feature vector set $X = \{(s^O, s^A, s_m, s_t)\}$ and the targets $Y = \{r_i^O\}$. A request r_i arises at time r_{it} with r_i^O the closest location from its origin. We then train a classifier to try and infer the next request location, given the current state.

TABLEAU 5.1 State features

Feature	Number of values
Time s_t	24+7
PRO s_l^O	$ L $
PRA s_l^A	$ L $
Motives s_m	$ M $

Step 2 : Location Classifier Training

A learning model F is defined and trained to predict which locations Y will most likely receive the following request based on the features X describing the problem's state and the fleet's previous activity during the day. We consider random forests, gradient boosting methods, and multilayer perceptrons for our classifiers. Details about the training are provided for the case study in Section 5.5.2. Once trained, the models give as an output a vector quantifying the probability of each location being the next receiving a request, $F(x) = \hat{y}$. It is then used to issue a repositioning decision to vehicles finishing a trip and idle for too long at an inactive location.

Step 3 : Algorithm for Vehicle Repositioning

The repositioning procedure is illustrated in Algorithm 3. It is called each time a decision must be made, i.e. when a request is issued or when an ambulance has finished its trip. The algorithm generates a prediction for the different locations with the trained model (line 9) and chooses locations to reposition the vehicles that have been idle for too long and the vehicle that finished its trip, if applicable (lines 11 to 13). The repositioning location is obtained by `chooseLocation`($\hat{y}, v, \mathcal{L}$) procedure (line 12), which is formalized in Algorithm 4. It is done for a vehicle $v \in V$, based on the score obtained for each position ($\hat{y}$) and on the current idling location of each vehicle ($\mathcal{L}$). The repositioning instructions are stored in $\mathcal{L}$ (line 12). The idling time for each vehicle is also updated (lines 13 and 15).

Algorithm 3: Fleet Repositioning

```

1 ▷ Pre :  $x$  is the current state, as defined in Step 1 (data preparation).
2    $F$  is the learned model (gives a score for each location from  $L$ ).
3    $V$  is the set of all vehicles.
4    $\rho$  is the vehicle that has finished its trip and is to be repositioned.
5    $\mathcal{L}[v]$  is the idling location of vehicle  $v$ .
6    $I[v]$  is the number of iterations during which  $v$  has been idle.
7    $\Theta$  is a threshold triggering a repositioning, if exceeded.
8
9    $\hat{y} := F(x)$  // Calling the trained model
10  for each  $v \in V$  do
11    if  $v = \rho \vee I[v] > \Theta$  then
12       $\mathcal{L}[v] := \text{chooseLocation}(\hat{y}, v, \mathcal{L})$  // Procedure defined in Algorithm 4
13       $I[v] := 0$ 
14    else
15       $I[v] := I[v] + 1$ 

```

The goal of Algorithm 4 is to find the closest empty location likely to receive a request. First, the number of vehicles present at each location is computed (line 11). Let $\hat{y} = \langle \hat{y}_l \rangle_{l \in L}$ be the score vector returned by the classifier for each location. The higher $\hat{y}_l$, the more confident the classifier is in having the subsequent request arising next to location l . Candidate locations Λ for repositioning are selected by taking all probable unfilled locations with a tolerance factor defined by $\gamma \in]0, 1[$ (line 15). A location is *unfilled* if it contains fewer vehicles than a given parameter $\tau \in \mathbb{N}^+$. For example, setting this parameter to 1 only allows picking an empty location until all locations have one idling vehicle. From those candidates, the vehicle is sent to the closest one (line 17). If all candidate locations are filled, we consider decreasingly probable locations until an unfilled one is found, which is done by ignoring the most probable

location and setting their score to 0 for the next iterations (line 19) as it did not yield a valid repositioning location. If all the locations are filled, the closest location is selected regardless of the number of vehicles idling there (line 21). This algorithm allows spreading the vehicles from the highest to lowest-valued locations, with the shortest distance being the decisive criterion for equally probable locations. We evaluate the practical use of the pipeline in a case study in the next section.

Algorithm 4: Selecting a Location for a Vehicle

```

1  ▷ Pre :  $\hat{y} = \langle \hat{y}_1, \dots, \hat{y}_L \rangle$  is the predictor output for each location.
2       $v$  is the vehicle to reposition.
3       $L$  is the set of all locations.
4       $\mathcal{L}[v]$  is the idling location of vehicle  $v$ .
5       $d[v, l]$  is the distance function from  $v$  to the location  $l$ .
6       $\gamma \in ]0, 1[$  is a prediction tolerance parameter.
7       $\tau \in \mathbb{N}^+$  is a filled location parameter.
8
9  chooseLocation( $\hat{y}, v, \mathcal{L}$ ) :
10     for each  $l \in L$  do
11          $C[l] := \left| \{v \in V \mid \mathcal{L}[v] = l\} \right|$            // Number of vehicles at each location
12      $l^* := \perp$                                            // Store the best location to reposition
13     while  $l^* = \perp \wedge \max(\hat{y}) > 0$  do
14          $k^* := \operatorname{argmax}(\hat{y})$            // Taking the index of location with highest score
15          $\Lambda := \{l \in L \mid \hat{y}_l \geq (1 - \gamma)\hat{y}_{k^*} \wedge C[l] < \tau\}$  // Filtering the locations
16         if  $\Lambda \neq \emptyset$  then
17              $l^* := \operatorname{argmin}_{l \in \Lambda} (d[v, l])$ 
18         else
19              $y_{k^*} := 0$ 
20     if  $l^* = \perp$  then
21          $l^* := \operatorname{argmin}_{l \in L} (d[v, l])$ 
22     return  $l^*$ 

```

5.5 Case Study : Ambulance Fleet in Belgium

This project is conducted in collaboration with a company to develop automated assistance for managing a fleet of ambulances. We consider a centralized dispatcher for the fleet that assigns identical ambulances to requests for medical transport and repositions idle ambulances to nearby hospitals. Even though requests arise randomly, patterns can be exploited as the requests tend to emerge more frequently in specific zones at certain times. Furthermore,

transport between hospitals might depend on why patients are being brought to the hospitals. Some interventions are short and patients will quickly be sent back home, some are long and require patients to spend the night at the hospital, and others might require an inter-hospital transfer. We use past trip data with geographical coordinates, time, and medical motives in our pipeline to study what impact a trained predictor has on overall fleet efficiency during real-world operating days. The pipeline is used to obtain a more reactive ambulance fleet by idling closer to oncoming requests; in particular, the company is interested in reducing the average reaction time, or the average reaction distance as a proxy. A study of the retraining of the predictor to improve over time and adapt to new data is presented in Appendix B.

5.5.1 Use of the Pipeline

We use collected data from 500 operating days to train our predictor. The data from past requests are transformed and rescaled for anonymity and better learning. The geographical coordinates of a request are mapped to the closest location (there are seven predefined hospitals used as waiting locations for ambulances), and the transport motive is encoded as a one-hot vector with predefined categories $M = \{\text{consultation}, \text{radiotherapy}, \dots\}$. As presented in Section 5.4, these encoded requests are used to build a state representation $x(t) = (s^O, s^A, s_m, s_t)$ at time t . The training set is composed of all previous requests, the state they arrived in, and the closest location to that request $\{x(r_{it}), r_i^O\}$.

5.5.2 Training Phase

Roughly 8000 real trips are available for training. We split this dataset into a training set and a validation set (85%-15%). We also keep 248 real trips (10 days of operations) for the test set. Three models have been implemented for the predictor : a *neural network*, *random forest*, and *gradient boosting*. The neural network is implemented with *PyTorch* and consists of a self-attention layer coupled to a two hidden-layer feed-forward neural networks with 250 and 150 neurons in each hidden layer, respectively. Training is carried out with the *Adam* optimizer [126], and we use an *early stopping* method to avoid overfitting. A binary cross-entropy loss is used for the cost function ($F_\theta(x) = \hat{y}$ trained to minimize $C(\theta) = -\frac{1}{N} \sum_{i=1}^N y_i \log(F_\theta(x_i)) + (1 - y_i) \log(1 - F_\theta(x_i))$ on batches of size N). Random forest and gradient boosting are implemented with *scikit-learn*. Each model is trained on a personal computer (i7-8750H 2.20GHz CPU) in less than 1 minute. The best values for the context vector parameters β and H are obtained by grid-search ($\beta = 0.75$ and $H = 10$). The evolution of the loss during training is reported in Figure 5.3(b) for the neural network. As comparison baselines we also integrate a random selection, and a constant heuristic *Most Frequent* always assigning the

location closest on average to the requests most frequently observed in the historical data. The accuracy (percentage of correctly predicted location) of the neural network is reported in Figure 5.3(a) and includes comparisons with gradient boosting, random, and the best constant heuristic *Most Frequent*.

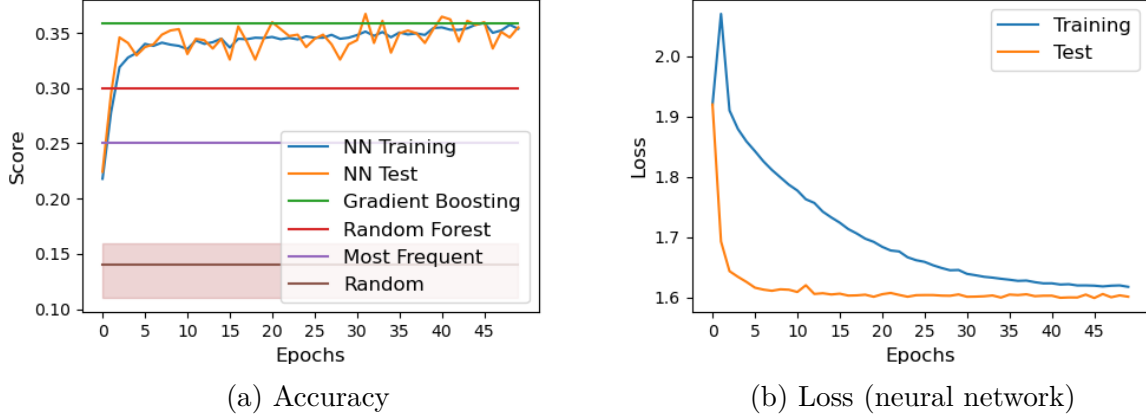


FIGURE 5.3 Statistics for the training phase.

5.5.3 Results : Location Prediction

Table 5.2 summarizes the main results of the different predictors, obtained with a 5-fold cross-validation method for predicting the next request, without any integration with the re-positioning algorithm. The accuracy and the average distance between the predicted locations and the requested positions are reported. We can observe that the three predictors outperform the heuristic, both in terms of accuracy and in terms of average distance, which is the result intended. Among the three predictors, *GradBoost* provides the best performance. We evaluate the impact of the improvement of the location prediction on the fleet performance in the next section.

TABLEAU 5.2 Results of the different predictors for the next request. The percentage of improvement compared to the most frequent heuristic is also proposed.

Predictor	Accuracy	Average distance to request (km)
Random	14%	13.1 ± 5.1
Most frequent	25%	8.58 ± 4.1
Random Forest	30%	8.53 ± 4.3 (-0%)
NN	35%	8.05 ± 4.3 (-6%)
GradBoost	36%	6.71 ± 4.1 (-21%)

5.5.4 Results : Ambulances Repositioning

Thanks to an *application programming interface* (API) provided by the partner company, we were able to obtain the travel time between each pair of ambulance depots and request position. With this information, we replayed *a posteriori* the requests of 10 historical days and re-positioned the ambulances with different policies : (1) positioning ambulances to the closest hospital after a trip (*Closest*), (2) sending ambulances to an empty hospital that was historically closest to the most requests (*Most frequent*, which is the previously seen heuristic restricted to empty locations), (3) the relocation procedure proposed in Algorithm 4 ($\gamma = 0.05$ and $\tau = 1$) with the three predictors (*NN*, *Random Forest*, and *GradBoost*). Results are illustrated in Figure 5.4, and Table 5.3 presents the numerical results. The *reaction distance* corresponds to the accumulated travel times of all the ambulances between their current position to the location of accepted requests, and the *total distance travelled* is the accumulated travel times of all the ambulances, including the repositioning operation.

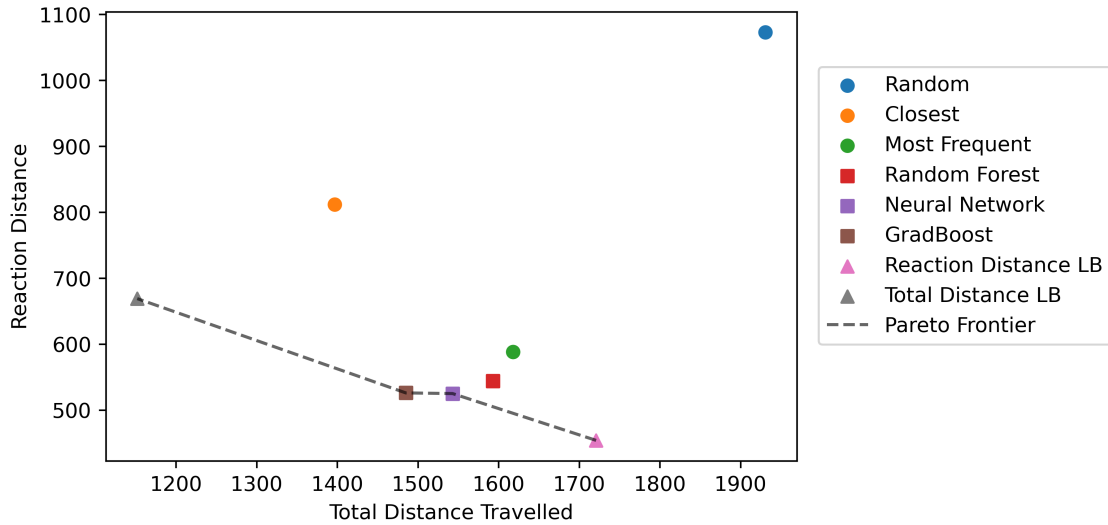


FIGURE 5.4 Distance to drive to requests in 10 simulated days (data in Table 5.3)

The results show that the learned predictors help ensure more precise placement of vehicles for better reactivity, improving the *Most frequent* strategy by 10% and the *closest* strategy by 35%. By design, *Most frequent* ensures reactivity but results in greater overall distance travelled by the fleet. This can be corrected by using the learned predictors, especially the gradient boosting method that brings the total distance travelled back to *Closest* levels. *Closest* yields a low total distance because most ambulance movements are reactive. The ambulances reposition to the closest hospital after fulfilling a trip and then go directly to the next arising request. This ensures that few detours are made, but the reaction distance might

TABLEAU 5.3 Distance (km) to drive to requests in 10 simulated days.

Predictor	Reaction distance (locations to requests)	Total distance traveled
Random	1073	1930.8
Closest	812.5	1412.1
Most frequent	587.6	1617.6
Random Forest	544.2	1593.4
NN	525.9	1552.6
GradBoost	526.3	1470.1
Reaction distance LB	453.8	1721.3
Total distance LB	669.2	1052.1

be large because of a lack of anticipation. As we replayed historical days, it is possible to compute the optimal repositioning decisions with perfect information, minimizing either the reaction distance or the total distance. Although not feasible in practice, it provides lower bounds on the distance that can be achieved. The results show that the learned predictors approach these theoretical lower bounds while keeping the total distance travelled relatively low.

We deployed our repositioning strategy on the partner company server and ran a live experiment on 20 new days. The prediction pipeline is used as described in the previous section, and we evaluate the impact it would have had compared to the real-life instructions given by the operators (partly *Closest* and partly *Most Frequent* strategies most of the time) by fetching data in the database and providing real-time predictions for repositioning. The results are summarized in Table 5.4 for 20 operational days and show that the reaction distance could have been diminished, by between 15% to 21%, while the total distance would have increased, by around 10%. Thus, the predictors succeed in improving fleet reactivity with minimal deterioration of the total distance; an intermediate strategy can be used if the fleet is averse to small increases in total distance.

TABLEAU 5.4 Distance (km) to drive to requests, averaged over 20 days.

Predictor	Reaction distance (locations to requests)	Total distance traveled
Dispatcher instructions	1002.7	1712.3
NN	787.6 (-21%)	1898.5 (+11%)
GradBoost	846.8 (-15%)	1882.0 (+10%)

5.6 Conclusion

We developed an adaptive learning pipeline to reposition fleet vehicles in a ride-hailing problem with centralized control. Prior data on trips and requests allowed us to train classifiers to determine the zone most likely to receive requests, and vehicles that need to be repositioned are dispatched according to the prediction and state of the fleet. Our simulated application to a real ambulance fleet in Belgium shows a theoretical improvement of more than 10% in reactivity for arising requests, and this result was confirmed by the implementation of our predictor in an industrial vehicle routing and fleet management program for ambulances.

CHAPITRE 6 APPRENTISSAGE AUTOMATIQUE POUR L'APPROXIMATION ET LA RÉOLUTION DE PROBLÈMES DE PLANIFICATION DE RENDEZ-VOUS DANS DES TOURNÉES DE VÉHICULES EN TEMPS RÉEL

6.1 Introduction

6.1.1 Motivation

La gestion conjointe d'emplois du temps et de tournées de véhicules est complexe et demande des algorithmes sophistiqués pour des problèmes de grande taille. Leur gestion et modification en temps réel est de fait un défi et demande une puissance de calcul et une modélisation compétitives. Nous nous intéressons à l'utilisation de l'apprentissage machine pour simplifier l'insertion de rendez-vous dans un emploi du temps optimisé en faisant un recours limité à des opérations de ré-optimisation mathématiques coûteuses et lentes durant un échange en temps réel avec un client potentiel. Nous développons une représentation sous forme d'hétérographes de solutions d'optimisation d'emploi du temps, ainsi qu'un modèle de réseau de neurones sur graphes permettant de prédire les meilleures insertion de rendez-vous dans ces graphes ; nous atteignons des précisions de l'ordre de 80% d'insertions optimales, permettant des propositions rapides quand une optimisation serait trop lente.

6.1.2 Préliminaires

La planification de tournées de véhicules représente une part importante des chaînes d'approvisionnement industrielles et de leurs coûts. Résoudre ces problèmes à l'optimalité, ou proche de l'optimalité, est donc un moyen important de réduire les coûts et gagner en efficacité globale. Ces problèmes sont néanmoins durs à résoudre, et un gain de temps et de complexité sur le calcul des tournées est un sujet de recherche actif depuis les débuts de la recherche opérationnelle [127]. Le problème de tournée de véhicules est également étudié pour ses nombreuses applications, par exemple pour les livraisons à domicile ou le ramassage scolaire, et ses multiples variantes, avec fenêtres de temps, capacités ou éléments dynamiques [128–130]. Néanmoins, la résolution de grand problèmes reste toujours un défi du fait de l'explosion combinatoire coûteuse en mémoire et en temps de calcul.

Une approche complémentaire à l'amélioration des solveurs, de la modélisation ou des heuristiques de résolution pour le VRP est le recours à l'apprentissage automatique. Si les premières approches essayaient essentiellement de générer directement des solutions [61–63], les

applications récentes se concentrent également sur guider la recherche à l'aide d'agents d'apprentissage [131], par exemple par l'estimation de la probabilité d'appartenance d'une arête au tour optimal ou le retrait d'arêtes peu prometteuses [66, 132], ou par choix heuristique de la direction de recherche [68–70, 133].

Certaines applications n'ont pas les mêmes objectifs, et nous nous intéressons au cas d'un opérateur souhaitant savoir instantanément si oui et quand un nouveau client peut être ajouté dans une journée de tournées optimisées, afin de pouvoir lui proposer des horaires de rendez-vous au téléphone en temps réel. Afin de simplifier ces problèmes et d'éviter des optimisations coûteuses, nous proposons l'utilisation de réseaux de neurones sur graphes afin d'apprendre d'optimisations passées pour construire des solutions partielles. Une application industrielle visant l'accélération de la prise de rendez-vous en temps réel est proposée.

Problème de tournée de véhicules

Nous nous intéressons pour la suite du chapitre à une formulation classique du problème de tournée de véhicules avec capacités et fenêtres de temps.

Paramètres :

$C = \{1, 2, \dots, n\}$: Ensemble des clients

$N = \{0, 1, \dots, n + 1\}$: Ensemble des clients et des dépôts de début et fin

K : Ensemble des véhicules

t_{ij} : Temps de trajet de i à j

Q_k : Capacité de k

q_i : Demande du client i

l_i : Durée de service de i

$[\tau_i^-, \tau_i^+]$: Fenêtre de temps de i

Variables :

$x_{ijk} \in \{0, 1\}$: Variable binaire indiquant si le véhicule k va de i à j

$s_{ik} \in \mathbb{R}$: Début du service de i par k

$y \in \mathbb{N}$: Nombre de clients servis

Fonction objectif :

Maximisation du nombre de clients servis, et minimisation du retour le plus tard :

$$\text{Maximise}_{x,y,s} \quad My - \max_{k \in K} s_{(n+1)k} \quad (6.1)$$

Contraintes :

y clients sont servis :

$$\sum_{i \in N} \sum_{j \in C} \sum_{k \in K} x_{ijk} = y$$

Chaque véhicule part et arrive aux dépôts respectifs :

$$\forall k \in K, \sum_{j \in N} x_{0jk} = 1, \sum_{i \in N} x_{i(n+1)k} = 1$$

Respect des capacités :

$$\forall k \in K, \sum_{i \in C} \sum_{j \in N} q_i x_{ijk} \leq Q_k$$

Un véhicule arrivant chez un client doit repartir :

$$\forall h, k \in C \times K, \sum_{i \in n} x_{ihk} = \sum_{j \in N} x_{hjk}$$

Respect des contraintes temporelles :

$$\forall i, j, k \in N^2 \times K, s_{ik} + t_{ij} + l_i - M(1 - x_{ijk}) \leq s_{jk}$$

$$\forall i, k \in C \times K, \tau_i^- \leq s_{ik} \leq \tau_i^+, s_{0k} = 0$$

Par soucis de simplicité tous les véhicules partent et reviennent au même dépôt, mais ils peuvent en pratique être différents pour chaque véhicule.

6.1.3 Approche

Nous nous intéressons à des résolutions successives du VRP avec ajout d'un client. Section 6.2 présente la représentation graphique du problème et de ses solutions, afin de définir une tâche d'apprentissage calculant la représentation cachée des noeuds du graphe pour en déduire les probabilités d'insertions fructueuses Section 6.3. Section 6.4 met en application l'approche chez une entreprise partenaire pour la prise de rendez-vous en temps réel.

6.2 Représentation d'une solution au VRP

Nous représentons le problème et une solution optimisée dans un graphe orienté contenant toutes les informations encodées dans la structure du graphe et ses paramètres sur les noeuds ou les arêtes. Nous détaillons ci-dessous la structure du graphe représentant le problème, ainsi que les vecteurs paramètre encodant des données du problème ainsi que la solution optimisée.

6.2.1 Structure du graphe

Soit $\mathcal{G}_I = (V, E)$ le graphe représentant le problème I . Soit A l'ensemble des rendez-vous à honorer sur une journée (*appointments*) et R l'ensemble des ressources (chauffeurs, techniciens, infirmières se déplaçant pour les rendez-vous). Les noeuds $V = ((v_a)_{a \in A}, (v_r)_{r \in R}, (v_t)_{t \in T})$ représentent les rendez-vous, les ressources et une discrétisation du temps. Les arêtes paramétrées $E = (E_{A-A}, E_{A-R}, E_{A-T}, E_{R-T}, E_{T-T})$ représentent les relations entre ces différents éléments. E_{A-A} indique les temps de trajet entre chaque lieu de rendez-vous, E_{A-R} quels rendez-vous peuvent être pris en charge par quelles ressources, E_{A-T} sur quels créneaux horaires un rendez-vous peut se dérouler, E_{R-T} sur quels créneaux horaires les différentes ressources travaillent, et E_{T-T} quels créneaux horaires sont consécutifs; voir par exemple Figure 6.1. Ainsi, $\forall a, r \in A \times R, e_{ar}$ n'existe que si r est compatible avec a , $\forall a, r \in A \times T, e_{at}$ n'existe que si t est inclus dans la fenêtre de temps de a , et $\forall r, t \in R \times T, e_{rt}$ n'existe que si r est disponible durant t . Les vecteurs paramètre, ou features, encodant plus d'information sur les noeuds et arêtes et constituant les vecteurs d'état initiaux pour l'inférence des GNN sont détaillés Section 6.2.2.

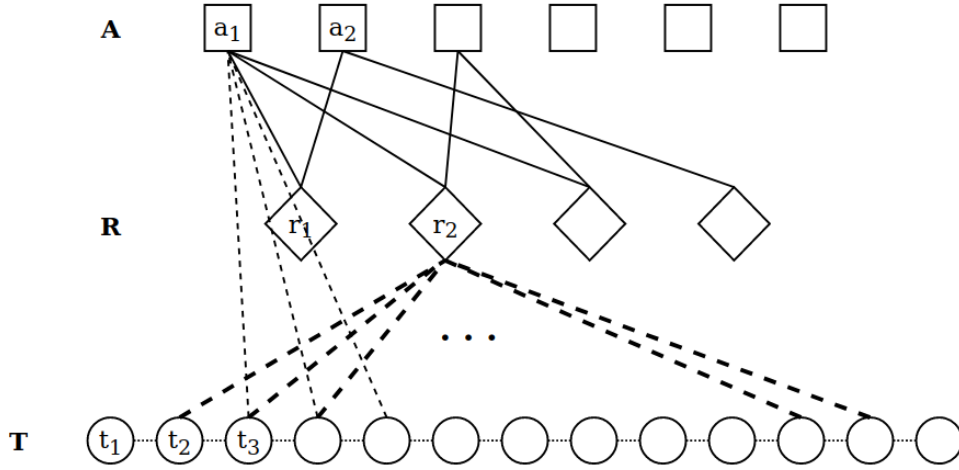


FIGURE 6.1 Exemple de graphe, illustrant des arêtes E_{A-R} , E_{A-T} , E_{R-T} , et E_{T-T} .

6.2.2 Vecteurs paramètre

Soit $h_v \in \mathbb{R}^d$, $v \in V$ et $h_e \in \mathbb{R}^d$, $e \in E$ les paramètres associés aux noeuds et arêtes du graphe. Les noeuds sont ainsi enrichis de features indiquant :

$v \in \dots$	h_v inclut
A	demande (q_i), durées de service (l_i)
R	capacité (Q_k)
T	progression de la journée, dans $[0,1]$

Idem pour les arêtes :

$e \in \dots$	h_e inclut
E_{A-A}	temps (t_{ij})
E_{A-R}	temps (t_{ij}), booléen indiquant si a est servi par r
E_{A-T}	booléen indiquant si a est prévu pendant t
E_{R-T}	booléen indiquant si r travaille pendant t

On donne Figure 6.2 un exemple de distances entre différents clients et dépôts initiaux. Les temps t_{ij} , durées de service l_i , capacités Q_k et demandes q_i sont en pratique normalisés entre $[0, 1]$. Les vecteurs paramètre h constituent les vecteurs d'état initiaux pour l'inférence GNN.

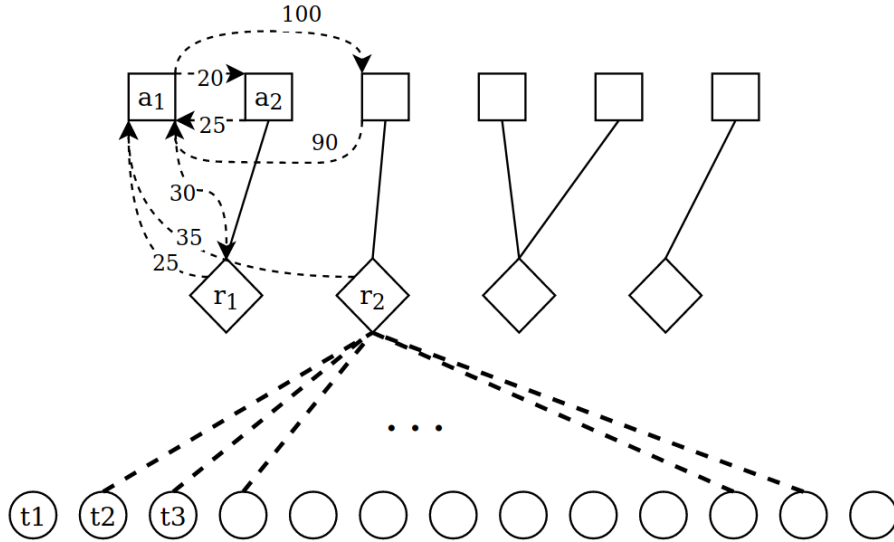


FIGURE 6.2 Exemple de graphe, illustrant des distances sur arêtes E_{A-R} , et E_{A-A} .

6.3 Apprentissage sur des solutions de VRP

Nous nous intéressons à l'insertion d'un rendez-vous dans un emploi du temps optimisé, et de façon équivalente à l'insertion d'un noeud α dans un graphe solution. Un GNN entraîné

sur des exemples d'insertion de rendez-vous effectués par résolutions de programmes de VRP peut estimer la probabilité d'une bonne insertion ; nous considérons pour cela une tâche de régression d'un score d'insertion sur les arêtes $(e_{\alpha-t})_{t \in T}$ qui représentent une attribution du rendez-vous α à un créneau horaire. Les exemples d'insertion de rendez-vous sont obtenus grâce à des données réelles et des résolutions du programme de VRP décrit Section 6.1.2.

6.3.1 Score d'insertion

Afin de guider l'estimation de bonnes insertions de rendez-vous, nous définissons un score quantifiant la qualité d'une insertion. Soit a un rendez-vous à insérer dans une journée d sur un créneau horaire $\tau \in T$ compris dans $[\tau_a^-, \tau_a^+]$, et $f^*(I_d, a, \tau)$ la valeur de la fonction objectif 6.1 pour cette insertion. En notant $s^+ = \max_{\tau} (f^*(I_d, a, \tau))$ et $s^- = \min_{\tau} (f^*(I_d, a, \tau))$ les valeurs minimum et maximum des insertions réalisables, on définit le score d'insertion normalisé $s_{a,\tau}$ de a au créneau τ :

$$s_{a,\tau} = \frac{f^*(I_d, a, \tau) - s^-}{s^+ - s^-}$$

On fixe $s_{a,\tau} = -1$ si l'insertion est irréalisable.

6.3.2 Ensemble d'entraînement

L'ensemble d'entraînement est constitué de journées d'opérations auxquelles nous retirons des rendez-vous, et apprenons à les réinsérer durant le meilleur créneau ; l'algorithme de création des exemples d'entraînement est détaillé Algorithme 5. Pour chaque jour présent dans les données, nous retirons une proportion de rendez-vous à réinsérer plus tard (A_d , Ligne 4), puis pour chaque réinsertion $a \in A_d$ nous retirons également une autre proportion de rendez-vous pour diversifier les situations rencontrées (R , Ligne 7). Un graphe représentant la solution du VRP sur les rendez-vous restants est construit (G_d^a , Ligne 9), puis pour tous les créneaux horaires couvrant la fenêtre de temps de a le score de l'insertion est calculé ($s[G_d^a, \tau_a]$, Ligne 11). Les graphes solution associés aux scores d'insertion des différentes demandes de A_d sont stockés pour entraînement ou test (Ligne 12).

Ceci permet de reproduire ce qu'il se passerait en pratique : un emploi du temps courant a été optimisé G_d^a , et il est réoptimisé avec l'arrivée d'un nouveau rendez-vous ; le score permet de quantifier quelle insertion horaire est la meilleure, et donc celle par laquelle il aurait été bon de commencer.

Algorithm 5 Création ensemble d'entraînement

```

1: Collection de  $\mathcal{I} = \{I_d\}$ , instances journalières de VRP
2: for  $d \in D$  do
3:    $\mathcal{S}_d \leftarrow \emptyset$ 
4:    $A_d \leftarrow 5$  rendez-vous de  $I_d$  sélectionnés aléatoirement
5:    $I_d \leftarrow I_d \setminus A_d$ 
6:   for  $a \in A_d$  do
7:      $R \leftarrow 10\%$  de rendez-vous de  $I_d$  sélectionnés aléatoirement
8:      $I_d \leftarrow I_d \setminus R$ 
9:      $G_d^a \leftarrow$  graphe solution de la résolution de  $I_d \setminus (R \cup A_d)$ 
10:    for  $\tau_a \in T \cup [\tau_a^-, \tau_a^+]$  do
11:       $s[G_d^a, \tau_a] \leftarrow$  score de la résolution de  $I_d \setminus (R \cup (A_d \setminus \{a\}))$ 
12:       $\mathcal{S}_d \leftarrow \mathcal{S}_d \cup \{G_d^a, \tau_a, s[G_d^a, \tau_a]\}$ 
13:    end for
14:  end for
15:   $\mathcal{S}_{\text{train}} \leftarrow \mathcal{S}_d$  ou  $\mathcal{S}_{\text{test}} \leftarrow \mathcal{S}_d$ 
16: end for

```

6.3.3 Apprentissage

F_1 est un réseau de neurones profond sur graphes à Λ couches calculant une représentation contextualisée de chaque noeud ($F_1(G) = H \in \mathbb{R}^{|V| \times d}$, $H = (h_v^\Lambda)_{v \in V}$) avec mécanisme d'attention et features sur les arêtes (EGAT, *Edge-Featured Graph Attention Network*), et F_2 un perceptron multicouche entièrement connecté (*fully-connected MLP*) défini pour une tâche de régression sur les représentations internes des noeuds de rendez-vous et de créneau horaire ($F_2(h_a^\Lambda, h_t^\Lambda) = s \in \mathbb{R}$).

Essentiellement, F_1 agrège les vecteurs paramètres voisins pondérés par un score α_{ij} d'attention :

$$\forall l \in [0, \Lambda - 1], \forall i \in V, h_i^{(l+1)} = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W_1^{(l)} h_j^{(l)} \right)$$

$$\alpha_{ij} = \frac{\text{LeakyReLU}([h_i || h_{ij} || h_j])}{\sum_{k \in \mathcal{N}(i)} \text{LeakyReLU}(W_1' [h_i || h_{ik} || h_k])}$$

avec $W_1 = (W_1', (W_1^{(l)})_{l \in [0, \Lambda-1]})$ poids du réseau à apprendre, afin de créer une représentation cachée des noeuds à chaque couche de convolution EGAT.

$$F_2(h_a^\Lambda, h_t^\Lambda) = \mathbf{MLP}_{W_2}([h_a^\Lambda || h_t^\Lambda])$$

On note $\mathbf{W} = (W_1, W_2)$ les poids ajustés par rétropropagation de gradient durant l'entraînement. Algorithme 6 détaille un passage d'entraînement sur les exemples générés Algorithme 5.

Algorithm 6 Epoch d'entraînement

```

1: for  $G, a, \tau, s_{a,\tau} \in \mathcal{S}_{\text{train}}$  do
2:    $H = F_1(G)$ 
3:    $s = F_2(h_a, h_\tau)$ 
4:    $\text{loss} = (s - s_{a,\tau})^2$ 
5:    $\mathbf{W} \leftarrow \mathbf{W} - \frac{d\text{loss}}{d\mathbf{W}}$ 
6: end for

```

6.4 Application aux tournées de techniciens

Cette approche est mise en pratique avec les données anonymisées d'un partenaire industriel gérant des techniciens inspecteurs en bâtiment. Sur trois années de données, 600 jours sont utilisés pour produire 3000 exemples d'entraînement répartis 80%-20% entre ensembles d'entraînement et de test. Les instances de VRP sont résolues avec un temps limite de dix secondes avec le solveur *routing* de OR-Tools [134]. L'entraînement sur ordinateur scientifique (AMD Rome 7532 @ 2.40 GHz 256M cache L3) demande 4 heures et 1Go de mémoire pour la génération de l'ensemble d'entraînement (graphes, résolutions de VRP successives et scores) et 1 heure d'entraînement du modèle pour 10 epochs.

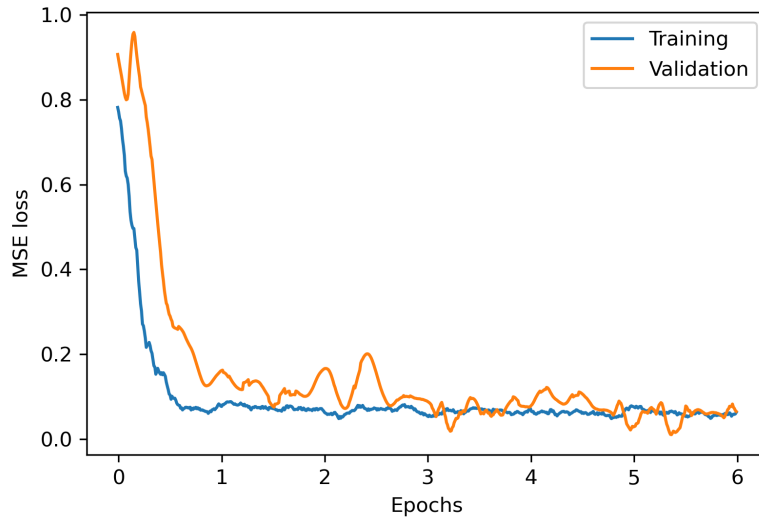


FIGURE 6.3 Courbes d'erreur quadratique moyenne $(s - s_{a,\tau})^2$ sur l'ensemble d'entraînement et de validation.

Le Tableau 6.1 présente les résultats de précision, soit la proportion de choix optimaux, et de score moyen des insertions proposées sur l’ensemble de test ; l’amélioration apportée par GNN est à chaque fois comparée à la ligne du dessus. Aléatoire choisit un créneau au hasard dans la fenêtre de temps, tandis que GNN choisit $\tau_{\text{GNN}} = \operatorname{argmax}_{\tau}(F_2(h_a, h_{\tau}))$. Si l’on souhaite étudier plusieurs possibilités d’insertions par optimisation, on peut également choisir plusieurs candidats : aléatoire top 3 est la meilleure insertion sur 3 choix aléatoires différents, et GNN top 3 est la meilleure insertion sur les 3 plus hauts scores proposés par GNN.

TABLEAU 6.1 Précision de l’insertion.

Méthode	Précision	Score moyen	Irréalizable
Aléatoire	35%	0.63	4%
GNN	81%	0.93 (+44%)	0%
Aléatoire top 3	72%	0.88	0%
GNN top 3	93%	0.98 (+12%)	0%

L’agent entraîné permet ainsi de souvent détecter l’insertion optimale avant optimisation, et la solution proposée n’est jamais irréalizable à défaut d’être optimale. Ces résultats confortent la possibilité d’une mise en service opérationnelle du prédicteur pour la prise de rendez-vous, et celui-ci permet de diviser par un facteur de 2 à 4 le temps de proposition d’horaires en fonction de la précision souhaitée.

6.5 Conclusion

Nous présentons une approche permettant d’estimer rapidement quelles sont les possibilités d’insertion d’un rendez-vous sur une journée déjà optimisée avec une précision satisfaisante, 80 à 90% d’insertions optimales sans propositions irréalisables, pour une mise en place opérationnelle chez notre partenaire industriel. Une adaptation à des données de tournées d’infirmières à domicile est également prévue. Plusieurs développements utilisant cette représentation graphique de VRP sont envisageables, comme l’adaptation à de plus grandes fenêtres de temps, l’association d’un rendez-vous à un technicien en plus du créneau horaire, ou l’utilisation d’un GNN pour prévoir différentes valeurs associées aux variables mathématiques du problème avant optimisation.

CHAPITRE 7 RELAXATION DE HELD ET KARP ET VOYAGEUR DE COMMERCE : GÉNÉRATION DE BORNE DUALE PAR APPRENTISSAGE PROFOND NON-SUPERVISÉ

7.1 Introduction

7.1.1 Motivation

Le problème du voyageur de commerce est connu tant pour la simplicité de son énonciation et son ubiquité dans les problèmes concrets et industriels que pour la difficulté de la résolution d’instances même de tailles modestes. Des approches naïves en terme d’énumération de solutions ou de programmation mathématique directe étant trop limitées, des techniques de séparation et évaluation (branch-and-bound) sont les plus souvent utilisées en branchant par exemple sur les variables fractionnaires d’une relaxation linéaires de la formulation en nombres entiers du problème. L’arbre de recherche ainsi créé croît exponentiellement avec la taille du problème mais peut être élagué au moyen de bornes d’optimisation. Nous exploitons la relaxation de Held et Karp pour générer des bornes duales fortes et entraînons un agent de génération de facteurs lagrangiens par apprentissage non-supervisé afin d’accélérer la génération de ces bornes et donc la résolution de problèmes de voyageur de commerce par branch-and-bound.

7.1.2 Revue de littérature

Le problème du voyageur de commerce, simple en apparence, est à l’origine de nombreux travaux de recherche. En raison de sa nature $\mathcal{NP}$ -difficile, de nombreuses approches ont été proposées pour le résoudre efficacement, allant des méthodes exactes aux techniques heuristiques. Held et Karp [135] proposent une relaxation offrant des bornes en pratique très fortes, qui sont par exemple utilisées de nos jours dans le performant solveur Concorde [13]. Un algorithme de branch-and-bound utilisant cette relaxation est également proposé [136], permettant notamment de fermer de nouvelles instances à l’époque de sa publication. Cette procédure faisant appel à des choix et heuristiques améliorables, nous proposons d’étudier comment améliorer son application en prédisant les facteurs lagrangiens de la relaxation au moyen d’un modèle prédictif, plutôt que de les calculer itérativement.

L’utilisation d’apprentissage conjointement aux techniques d’optimisation classiques est un sujet de recherche actif, par exemple dans la sélection de variable [69, 70] ou de noeud [68] dans un algorithme de branch-and-bound ; nous tournons le lecteur intéressé vers la revue de

littérature de Bengio et al. [137] décrivant comment les problèmes d’optimisation génériques peuvent être vus comme des points de données pour l’apprentissage d’une tâche donnée, posant la question des distributions pertinentes de problème à utiliser. Les réseaux de neurones sur graphes sont également une composante importante du succès de l’application de l’apprentissage profond à l’optimisation combinatoire [57]. Ils permettent d’extraire des représentations cachées riches par agrégations successives des poids de noeuds voisins dans un graphe, sur lesquels de nombreux problèmes combinatoires sont définis. Les réseaux de neurones sur graphes fournissent donc une représentation interne sur laquelle il est possible de raisonner, au niveau du graphe ou des noeuds, pour de la prise de décision. Nous étudions ainsi comment utiliser ces modèles pour apprendre à générer des facteurs lagrangiens adaptés pour chaque noeud du graphe afin d’appliquer la relaxation de Held et Karp, ainsi qu’une application au sein d’un algorithme de branch-and-bound pour la résolution complète du problème de voyageur de commerce.

7.1.3 Approche

Nous proposons dans la suite de ce chapitre l’utilisation d’un modèle d’apprentissage profond génératif pour amorcer plus rapidement un processus de génération de borne au sein d’un algorithme de branch-and-bound. Section 7.2 rappelle le principe de la relaxation de Held et Karp pour le voyageur de commerce. Sur cette base, Section 7.3 décrit l’approche d’apprentissage proposée pour la génération de bornes par apprentissage non-supervisé sur les facteurs langrangiens de la relaxation de Held et Karp, et finalement Section 7.4 décrit l’entraînement et l’intégration de la génération de bornes duales dans un algorithme de branch-and-bound pour évaluer leur impact sur la résolution d’instances du voyageur de commerce.

7.2 Relaxation lagrangienne de Held et Karp

Le problème du voyageur de commerce est difficile à résoudre et la résolution de grandes instances demande des approches sophistiquées face à l’explosion combinatoire de l’espace des solutions. Dans les méthodes de *branch-and-bound*, les bornes d’optimisation permettent d’élaguer l’arbre de recherche afin de finir la recherche et prouver l’optimalité sans avoir à l’explorer entièrement, ce qui serait en pratique impossible. La relaxation de Held et Karp [135] permet de générer une borne duale forte pouvant ainsi bénéficier à un solveur exploitant les bornes d’optimisation. Soit $G = (V, E)$ un graphe complet, on définit un *1-arbre minimum* comme un arbre couvrant minimum sur $G \setminus \{1\}$ auquel on ajoute $\{1\}$ et deux arêtes les moins coûteuses reliant $\{1\}$ à l’arbre. Ce 1-arbre peut être obtenu par la résolution d’un programme en nombres entiers Lignes 7.1 à 7.5 mais est en pratique facile à obtenir en appliquant

l'algorithme de Kruskal pour l'arbre couvrant minimum. $\delta(v)$ désigne l'ensemble des arêtes contenant v . On note c_e le coût des arêtes, x_e est la variable de décision indiquant si e est inclus dans le 1-arbre ou non.

$$\min \sum_{e \in E} c_e x_e \quad (7.1)$$

$$s.t. \sum_{e \in \delta(1)} x_e = 2 \quad (7.2)$$

$$\sum_{e \in E} x_e = |V| \quad (7.3)$$

$$\sum_{i,j \in S, i < j} x_{(i,j)} \leq |S| - 1, \quad \forall S \subset V \setminus \{1\}, |S| \geq 3 \quad (7.4)$$

$$x_e \in \{0, 1\}, \quad \forall e \in E \quad (7.5)$$

On remarque qu'un tour dans G est un 1-arbre, la construction d'un 1-arbre fournit donc une borne inférieure sur la valeur du tour optimal. En particulier, si le 1-arbre minimum est un tour, il est un tour minimum. En général le 1-arbre minimum ne sera pas un tour et fournira une borne éloignée. Pour obtenir un tour, on peut ajouter la contrainte

$$\forall v \in V, \sum_{e \in \delta(v)} x_e = 2 \quad (7.6)$$

mais le problème revient alors à trouver le cycle hamiltonien de coût minimum. On peut alors plutôt rapprocher le 1-arbre d'un tour en considérant la relaxation lagrangienne de cette contrainte, qui avec les facteurs lagrangiens θ_v , aussi appelés pénalités, pour chaque noeud du graphe fournit le nouvel objectif pénalisant les degrés différents de 2 :

$$\min \sum_{e \in E} c_e x_e - \sum_{v \in V} \theta_v (2 - \sum_{e \in \delta(v)} x_e) \quad (7.7)$$

qu'on peut aussi écrire

$$\min \sum_{e \in E} (c_e + \theta_{e_1} + \theta_{e_2}) x_e - 2 \sum_{v \in V} \theta_v \quad (7.8)$$

avec e_1 et e_2 les noeuds de l'arête e .

Comme la transformation $c'_{ij} = c_{ij} + \theta_i + \theta_j$ modifie le 1-arbre minimum mais pas le cycle hamiltonien minimum, on peut générer des bornes inférieures sur le problème grâce à l'Équation 7.8, et l'enjeu repose en trouver de bons facteurs lagrangiens pour obtenir une bonne

relaxation. On cherche à se rapprocher de la meilleure borne

$$\max_{\theta} \left[\min_x \sum_{e \in E} c_e x_e - \sum_{v \in V} \theta_v (2 - \sum_{e \in \delta(v)} x_e) \right]$$

La Figure 7.1 présente la relaxation de Held et Karp pour un graphe dont la valeur du tour optimal est 62. Le 1-arbre avec pénalités (graphe (2), facteurs lagrangiens en gras) a une valeur de 82, mais a décalé la valeur optimale de deux fois la somme des facteurs lagrangiens, soit 24 ; on obtient donc une borne plus proche que sans pénalités.

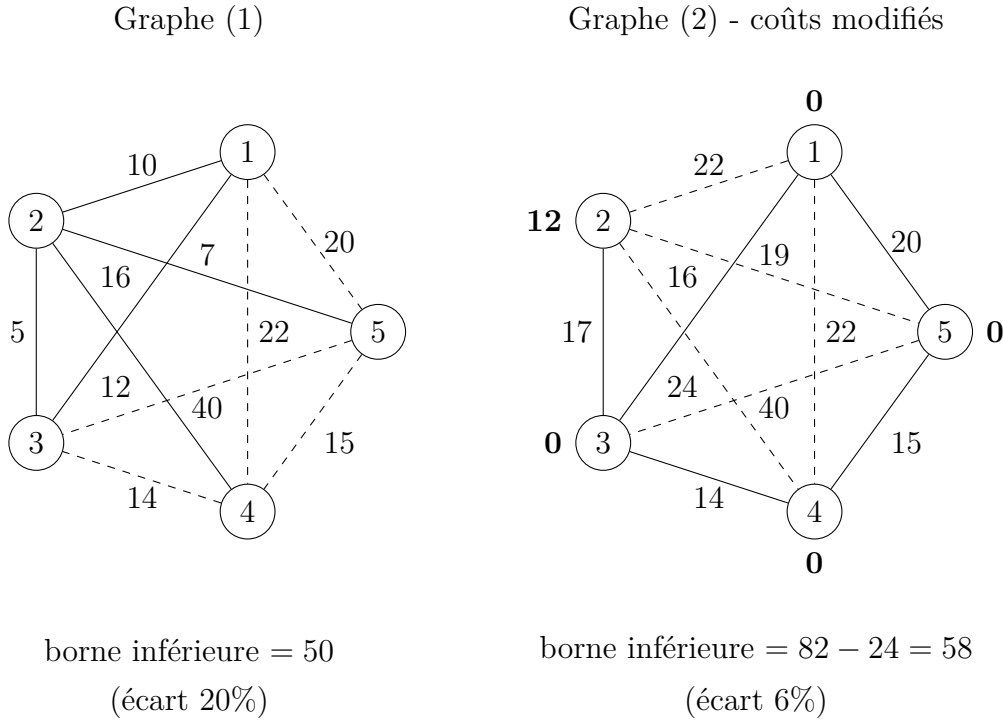


FIGURE 7.1 Relaxation de Held et Karp sans (1) et avec (2) pénalités.

Un processus itératif [136] permet d'ajuster les facteurs lagrangiens θ pour rapprocher le 1-arbre d'un tour en pénalisant les noeuds dont le degré n'est pas 2, et ainsi rapprocher la borne inférieure obtenue de l'optimal. Cette approche est généralement incluse dans un algorithme de *branch-and-bound* pour lequel l'arbre de recherche généré sera réduit grâce aux fortes bornes duales obtenues. Si calculer un 1-arbre est peu coûteux en pratique, l'ajustement itératif de θ peut l'être et nous proposons ainsi d'estimer directement θ à partir de la structure du graphe par apprentissage automatique afin d'accélérer l'obtention d'une borne inférieure forte.

7.3 Apprentissage non-supervisé pour Held et Karp

Soit $G = (V, E)$ un graphe connecté avec coûts c_{ij} pour $(i, j) \in E$ et $N = |V|$ noeuds. La génération de θ par un modèle d'apprentissage profond permet de calculer un 1-arbre sur G et d'utiliser le degré des noeuds dans le 1-arbre comme gradient pour la rétropropagation afin d'apprendre à générer des facteurs lagrangiens rapprochant le 1-arbre résultant d'un tour.

7.3.1 Génération des facteurs lagrangiens

Soit $h_v \in \mathbb{R}^d$, $v \in V$ et $h_e \in \mathbb{R}^{d'}$, $e \in E$ les features associées aux noeuds et arêtes du graphe. Les noeuds sont enrichis de paramètres indiquant (1) la distance moyenne de tous les autres noeuds du graphe, (2) la distance au noeud du graphe le plus proche, (3) leur degré dans le 1-arbre minimum sans facteurs lagrangiens, (4) un booléen indiquant le noeud $\{1\}$. Les arêtes contiennent des paramètres indiquant (1) le coût c_{ij} , ou distance, associé à cette arête, (2) un booléen indiquant si l'arête est interdite dans le tour, (3) un booléen indiquant si l'arête est obligatoire dans le tour.

Grâce à un modèle d'apprentissage profond sur graphes augmenté d'un mécanisme d'attention $f^W : \mathbb{R}^{N \times d} \rightarrow \mathbb{R}^N$, on associe à chaque noeud v un facteur θ_v pour la relaxation de Held et Karp. Équation 7.10 présente l'étape d'agrégation et fusionnement des features voisines pour le calcul de représentations cachées successives, jusqu'à génération de la représentation finale $h_v^{(\Lambda)}$ utilisée en entrée d'un réseau de neurones entièrement connecté pour la génération du facteur lagrangien (Équation 7.9). Le réseau de neurones est enrichi d'un mécanisme d'attention pour une plus grande expressivité (Équation 7.10 et 7.11).

$$\forall v \in V, \theta_v = \text{MLP}_{W_3}(h_v^{(\Lambda)}) \quad (7.9)$$

$$\forall l \in [0, \Lambda - 1], \forall v \in V, h_v^{(l+1)} = \sigma \left(\sum_{u \in \mathcal{N}(v)} \alpha_{uv} W_1^{(l)} h_u^{(l)} \right) \quad (7.10)$$

$$\alpha_{vu} = \frac{\text{LeakyReLU}(W_2[h_v || h_{(v,u)} || h_u])}{\sum_{k \in \mathcal{N}(v)} \text{LeakyReLU}(W_2[h_v || h_{(v,k)} || h_k])} \quad (7.11)$$

7.3.2 Dérivation pour la rétropropagation de gradient

Le programme linéaire en nombres entiers ci-dessus, avec la fonction objectif 7.1 remplacée par celle avec coûts modifiés 7.8, permet de calculer le 1-arbre minimum $X(\theta) = X(f^W(G))$.

La borne inférieure est donnée par

$$g(X(\theta)) = (C + \Theta)^\top X(\theta) - 2 \sum_{i=1}^n \theta_i$$

avec $\Theta_e = \theta_{e_1} + \theta_{e_2}$. La dérivée locale de cette borne nous donne donc

$$\frac{dg(X)}{d\theta_i} = \sum_{e \in \delta(i)} x_e - 2$$

et permet d'entraîner un prédicteur sur graphes comme suit pour maximiser la borne générée à partir de $\theta = f^W(G)$. La génération de facteurs lagrangiens est donc vue comme une tâche d'apprentissage non-supervisé cherchant à maximiser directement la borne correspondante aux facteurs. Le gradient ci-dessus correspond à l'intuition de vouloir corriger les facteurs correspondant aux noeuds de degré différent de 2 dans le 1-arbre résultant.

Algorithm 7 Epoch d'entraînement non-supervisé pour la génération de θ

```

1: Données d'entraînement :  $G$ 
2: for  $g \in G$  do
3:    $\theta_g = f^W(g)$ 
4:   Calcul du 1-arbre  $X(\theta_g)$ 
5:    $W \leftarrow W + \sum_{v \in V_g} \frac{d\theta_i}{dW} \times \frac{dg(X)}{d\theta_i}$ 
6: end for
7: return GNN entraîné  $f^W$ 

```

7.3.3 Génération des données

L'Algorithme 7 permet d'entraîner un agent pour la génération de facteurs lagrangiens pour la relaxation de Held et Karp sur des graphes de la distribution de G . Nous construisons différents types de graphes d'entraînement en fonction de l'utilisation qu'il sera faite du GNN. En effet s'il est utilisé à chaque noeud d'un algorithme de branch-and-bound branchant sur les arêtes du graphe, où certaines arêtes seront donc obligatoires ou exclues pour le tour recherché, l'entraînement doit présenter des graphes résultant d'un processus de branchement similaire. Nous limitons la profondeur jusqu'à laquelle le GNN est utilisé car s'il permet de générer de bonnes bornes, l'inférence GNN peut être coûteuse et moins utile dans un stade déjà avancé de l'arbre de recherche. Soit Q_g l'ensemble des noeuds de branch-and-bound explorés pour la résolution du voyageur de commerce sur le graphe g , et g_q le graphe considéré au noeud $q \in Q_g$. g_q est le graphe du problème enrichi de features représentant les informations récoltées et décisions prises durant le branch-and-bound comme décrit Section 7.3.1, notamment les arêtes forcées ou retirées du graphe. Soit k la profondeur maximale à laquelle

nous souhaitons utiliser un GNN entraîné ; nous construisons G_k un ensemble d'entraînement représentatif des graphes rencontrés par GNN_k grâce à l'Algorithme 8.

Algorithm 8 Génération de l'ensemble d'entraînement pour GNN_k

```

1:  $G_k = \emptyset$ 
2: Ensemble de graphes complets aléatoires :  $G$ 
3: for  $g \in G$  do
4:   for  $q \in Q_g$  do
5:     if profondeur( $q$ )  $\leq k$  then
6:        $G_k \leftarrow G_k \cup \{g_q\}$ 
7:     end if
8:   end for
9: end for
10: return  $G_k$ 

```

7.4 Application

Nous entraînons des générateurs de facteurs lagrangiens pour la relaxation de Held et Karp et testons les gains obtenus dans un algorithme de branch-and-bound classique.

7.4.1 Entraînement

Nous générons 100 graphes aléatoires géométriques, dont les coordonnées des noeuds sont tirées uniformément dans $[0, 1]$, de taille $n = \{50, 80, 100\}$ et produisons avec l'Algorithme 5 les ensembles d'entraînement G_k^n avec $k = 10$, valeur déterminée empiriquement. Soit f^W un réseau de neurones profond sur graphes avec mécanisme d'attention (EGAT, *Edge-Featured Graph Attention Network*) calculant une représentation vectorielle cachée de chaque noeud par agrégations puis concaténations successives des noeuds voisins. L'Algorithme 7 utilisé avec mini-lots (*mini-batches*) permet d'entraîner en quelques heures sur un processeur i7-8750H 2.20GHz les GNN associés aux ensembles d'entraînement générés. Les graphes et le calcul des arbres couvrants minimum passent par le module *NetworkX* et le modèle d'apprentissage EGAT est défini avec *Deep Graph Library* [138]. On retrouve les bornes générées sur les ensembles d'entraînement et de validation durant l'entraînement Figure 7.2.

UB est la valeur du tour optimal, que l'on souhaite approcher avec les autres courbes. *Unsupervised learning* est l'approche présentée ci-dessus. On remarque qu'elle performe mieux qu'une approche plus classique *Supervised learning* qui essaye d'imiter de bons facteurs générés par itérations successives avec l'approche classique de Held et Karp [135] ; ceux-ci sont également utilisés pour générer la borne *HK thetas* qui est meilleure que *Unsupervised lear-*

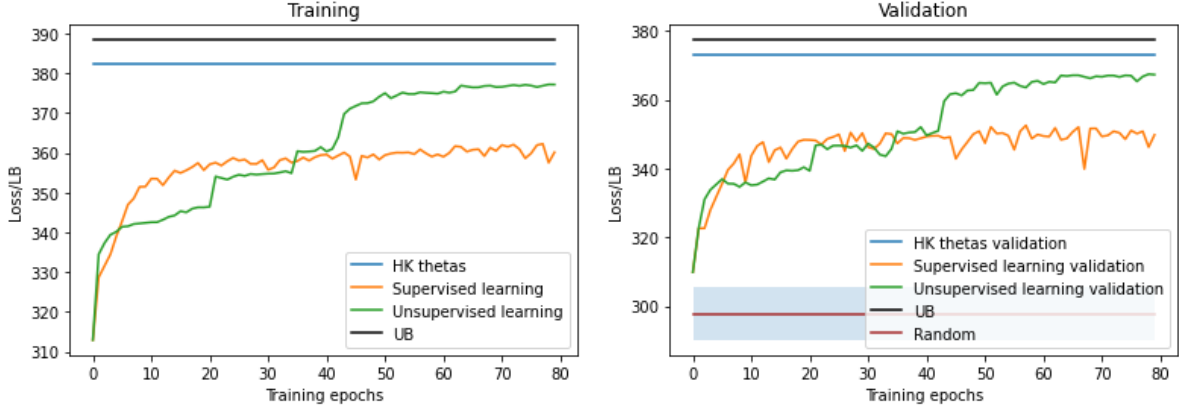


FIGURE 7.2 Entraînement pour la génération de facteurs lagrangiens sur des graphes G_1^{50} .

ning mais plus longue à obtenir. Une comparaison entre les bornes *Unsupervised learning* et *HK thetas* dans un algorithme de branch-and-bound est proposée Section 7.4.2. Finalement, *Random* qui utilise des facteurs lagrangiens aléatoires permet de vérifier que les facteurs générés par apprentissage sont effectivement performants.

7.4.2 Intégration branch-and-bound

Nous considérons un algorithme de branch-and-bound utilisant le relaxation de Held et Karp à chaque noeud pour filtrer les arêtes en fonction d'un calcul de coût de remplacement ou d'insertion; une borne duale forte permet un meilleur élagage des noeuds de branch-and-bound et un meilleur filtrage des arêtes, résultant en une résolution plus rapide.

Pour la relaxation de Held et Karp à chaque noeud de l'exploration branch-and-bound, *HK* désigne ici la pratique habituelle de fixer à 0 les facteurs lagrangiens avant de les adapter itérativement en fonction du 1-arbre généré, et *GNN* le départ avec facteurs générés par le modèle profond sur graphes puis adaptés itérativement de la même manière que *HK*. GNN_k utilise *GNN* jusqu'à profondeur k , puis *HK* pour la suite des noeuds du branch-and-bound. Le temps de calcul, le nombre de noeuds explorés, et l'intégrale primale-duale sont reportés dans les Tableaux 7.1 à 7.3 pour des résolutions de 50 graphes aléatoires pour le TSP. Les Figures 7.3 à 7.6 illustrent ces résultats, avec les points au-dessus de la courbes résolus plus efficacement par GNN_{10} .

Bien que quelques graphes difficiles semblent bénéficier de l'utilisation de GNN_{10} (Figure 7.3), il n'y a pas d'approche vraiment dominante pour les graphes faciles et le coût initial de chargement du GNN est donc pénalisant pour la majorité des graphes. GNN_{∞} n'est pas reporté par la suite du fait du temps de calcul prohibitif : la génération de facteurs lagrangiens

TABLEAU 7.1 Différentes générations de facteurs lagrangiens pour la relaxation de Held et Karp au sein d'un algorithme de branch-and-bound ($n = 50$).

	Temps (s)	Noeuds de branch-and-bound	Intégrale primale-duale
HK	18	22.2k	68.8k
Entraînement G_{10}^{50}			
GNN_1	17 (-6%)	19.7k (-11%)	62.2k (-9%)
GNN_{10}	14 (-22%)	14.5k (-35%)	44.0k (-36%)
GNN_{∞}	64 (+253%)	25.2k (+13%)	86.7k (+25%)

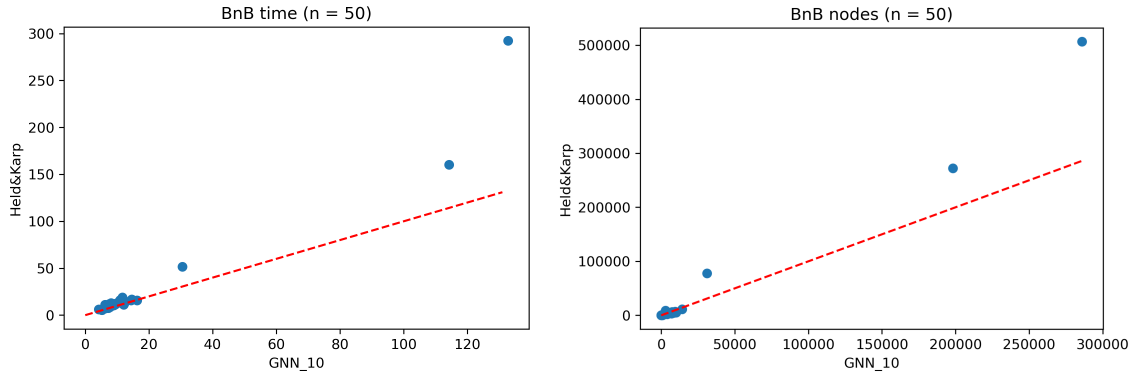


FIGURE 7.3 Comparaison du nombre de noeuds et du temps de résolution du TSP sur des graphes de taille 50 entre les relaxations HK et GNN_{10} .

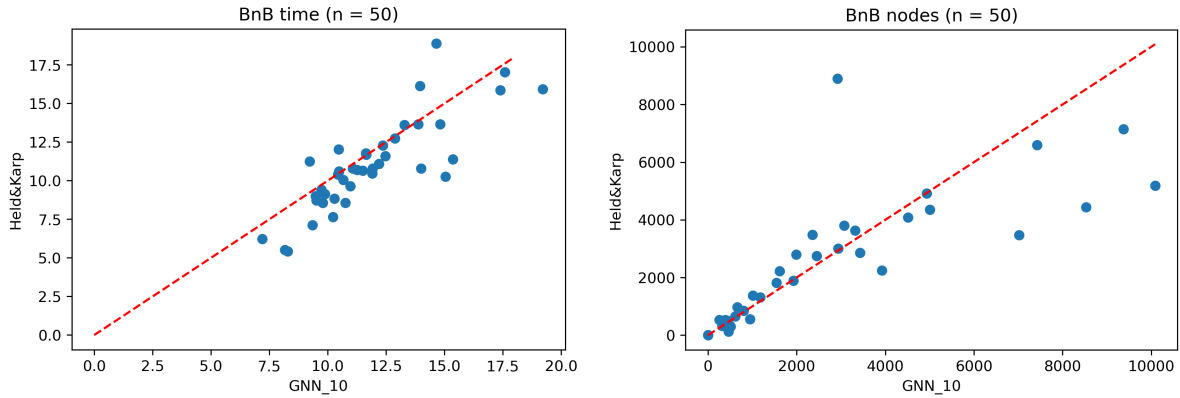


FIGURE 7.4 Comparaison du nombre de noeuds et du temps de résolution du TSP sur des graphes de taille 50 entre les relaxations HK et GNN_{10} (points extrêmes retirés).

pour des graphes différents de la distribution vue pendant l'entraînement est peu performante et trop pénalisante.

Bien que GNN_{10} demande un coût temporel initial dans le chargement du GNN entraîné et la construction du graphe d'entrée pour le GNN, la réduction du nombre de noeuds explorés

TABLEAU 7.2 Différentes générations de facteurs lagrangiens pour la relaxation de Held et Karp au sein d'un algorithme de branch-and-bound ($n = 80$).

	Temps (s)	Noeuds de branch-and-bound	Intégrale primale-duale
HK	185	190.1k	505k
Entraînement G_{10}^{80}			
GNN ₁	182 (-2%)	174k (-8%)	470k (-6%)
GNN ₁₀	154 (-17%)	137.9k (-27%)	361k (-28%)

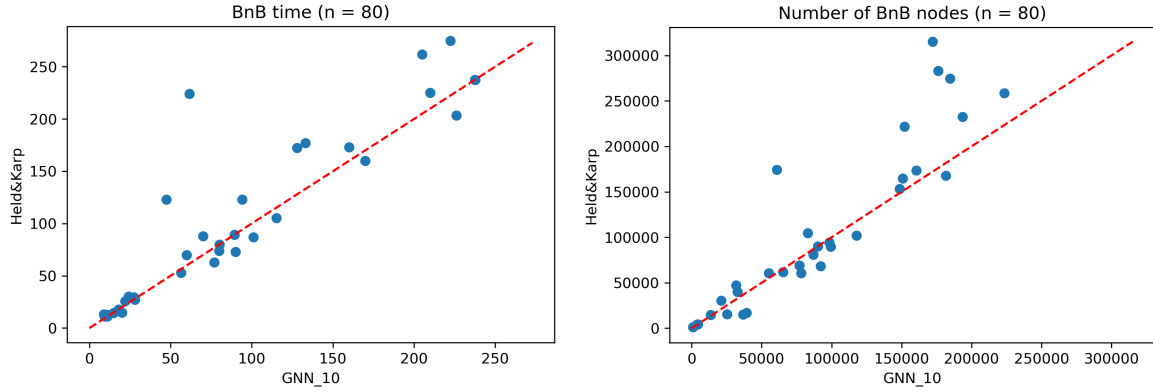


FIGURE 7.5 Comparaison du nombre de noeuds et du temps de résolution du TSP sur des graphes de taille 80 entre les relaxations HK et GNN_{10} .

TABLEAU 7.3 Différentes générations de facteurs lagrangiens pour la relaxation de Held et Karp au sein d'un algorithme de branch-and-bound ($n = 100$).

	Temps (s)	Noeuds de branch-and-bound	Intégrale primale-duale
HK	366	430.9k	1577k
Entraînement G_{10}^{100}			
GNN ₁	347 (-5%)	391.4k (-9%)	1417k (-10%)
GNN ₁₀	299 (-18%)	336.6k (-22%)	1228k (-22%)

grâce aux meilleures bornes permet de compenser ce coût et d'améliorer le temps de résolution total d'environ 20% pour les instances de tailles 50 à 100. S'il n'y a pas d'approche dominante pour tous les graphes, GNN_{10} permet tout de même une amélioration en moyenne.

7.5 Conclusion

La génération par apprentissage profond de solution à un problème d'optimisation pose des problèmes de faisabilité ou de garanties de qualité, résultant en une approche difficilement uti-

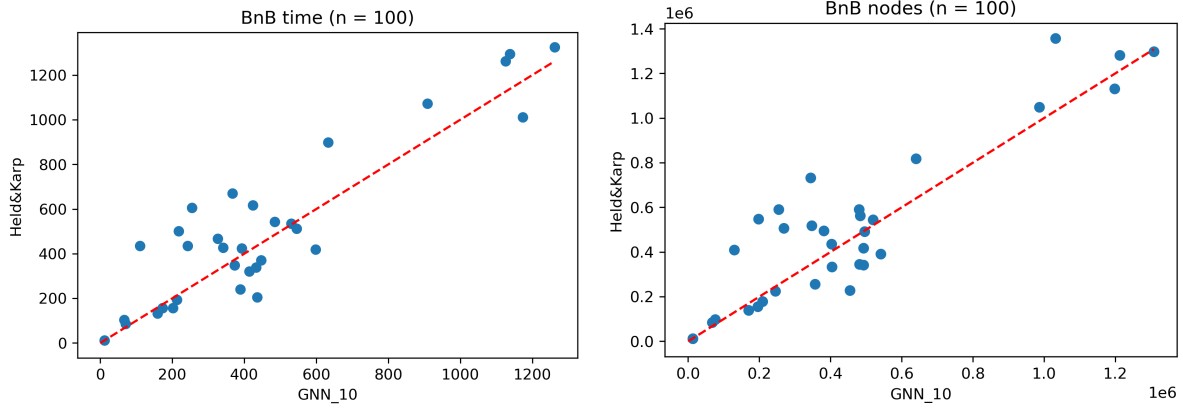


FIGURE 7.6 Comparaison du nombre de noeuds et du temps de résolution du TSP sur des graphes de taille 100 entre les relaxations HK et GNN_{10} .

lisible en pratique. Néanmoins, la résolution de problèmes guidée par apprentissage présente des performances bénéfiques sans renoncer aux garanties de l'optimisation. Nous démontrons dans ce projet l'intérêt de l'apprentissage non-supervisé pour la génération de facteurs lagrangiens pour accélérer et renforcer l'obtention de bornes duales pour le voyageur de commerce. L'insertion d'un modèle d'apprentissage générateur de facteurs lagrangiens pour la relaxation de Held et Karp dans un branch-and-bound classique permet ainsi une réduction de 22 à 35% du nombre de noeuds explorés. Les gains en temps de résolution sont plus délicats à obtenir du fait du coût des passes d'inférence GNN, mais des réductions de 18 à 22% du temps de calcul sont observées sur les différentes instances testées. Le problème du voyageur de commerce est un paradigme fondateur pour beaucoup d'applications, mais cette approche pourrait être également plus généralement employée dans diverses utilisations de relaxation lagrangienne.

CHAPITRE 8 DISCUSSION GÉNÉRALE

Cette thèse propose plusieurs applications de l'apprentissage automatique pour des problèmes de recherche opérationnelle, dont plusieurs enseignements peuvent être tirés. Nous développons ici les défis rencontrés, les approches retenues ainsi que les connexions entre ces différents projets.

8.1 Apprentissage automatique et faisabilité

Une solution proposée par optimisation mathématique à un problème bien défini a pour avantage d'être directement réalisable. En effet, les contraintes associées au problème, qui restreignent les variables et donc la recherche, représentent des limitations réelles ou désirées sur la solution qui doit être mise en pratique. Au contraire, la sortie d'un réseau de neurones entièrement connecté classique ne répond généralement à aucune contrainte autre que la forme qui lui est donnée par l'architecture du modèle. Si elle est entraînée sur des exemples, il n'y a pas de garanties sur la sortie produite, que l'on soit hors de la distribution des exemples rencontrés ou non. Cette liberté qui existe par définition et qui peut être un avantage pour l'apprentissage profond demande un effort de définition et d'interfaçage lorsque ces modèles sont utilisés dans un contexte pour lequel faisabilité et optimalité sont attendues. Une solution entièrement générée par un modèle d'apprentissage demandera par exemple généralement un effort de correction ou reconstruction de la solution pour la rendre réalisable. Ce problème est évitable par exemple en utilisant des modèles d'apprentissage uniquement comme une partie de l'approche de résolution, sans que leur impact puisse être bloquant en cas de génération de sortie inattendue ou indésirable. Ainsi, le projet de repositionnement d'ambulances utilise un modèle prédictif servant à donner du relief à la distribution géographique des requêtes futures afin d'être intégré dans une approche plus classique de prise de décision pour le repositionnement. Le projet d'insertion de rendez-vous dans un VRP utilise un modèle prédictif directement pour l'insertion ; si celle-ci pourrait être irréalisable, ce n'est pas le cas en pratique et les prédictions sont utilisées en entrée d'un solveur mathématique pour vérification et production de nouveaux emplois du temps a posteriori. L'intégration de modèles d'apprentissage comme heuristiques au sein d'un algorithme de branch-and-bound, Chapitres 4 et 7, tire quant à elle parti de l'impossibilité d'un choix irréalisable par l'heuristique car le choix d'ordre des couches du d'un diagramme de décision est toujours possible pour la construction du diagramme, et le choix de valeur de facteurs lagrangiens n'est pas contraint pour la relaxation du problème de voyageur de commerce.

8.2 Modélisation pour l'apprentissage et l'optimisation

Une autre considération dans l'utilisation jointe de l'apprentissage profond et de l'optimisation est la modélisation du problème ou de l'état du problème. En effet, si l'optimisation travaille avec une structure de définition rigide basée sur une fonction objectif, des variables de décision, des contraintes et des domaines de définition, l'apprentissage profond s'intéresse généralement à des données moins structurées. Nous avons fait le choix dans la majorité des projets présentés de recourir aux réseaux de neurones sur graphes qui, contrairement aux réseaux de neurones classiques intervenant sur des vecteurs ou matrices statiques, définissent des schémas d'agrégation et de concaténation permettant de traiter des graphes de tailles et formes arbitraires. La représentation sous forme de graphe permet de considérer des entités paramétrées et leurs relations, sans avoir à les ordonner arbitrairement. Ces qualités sont particulièrement adaptées à la représentation de nombreux problèmes mathématiques, au contraire par exemple d'une image ou d'un enregistrement audio pour lesquels la cohérence spatiale ou temporelle est indispensable à la reconnaissance de motifs. Ainsi, le problème de stable maximum traité Chapitre 4 n'a pas besoin de traduction intermédiaire mais est immédiatement transmissible à un GNN puisque le problème est entièrement défini par son graphe d'origine. De même, le problème du voyageur de commerce Chapitre 7 est directement représentable par un graphe, qui est amené à évoluer durant la recherche par branch-and-bound en fonction des décisions prises. Le Chapitre 6 est lui le fruit d'un travail de représentation d'un problème de VRP et de ses solutions par un graphe riche en informations afin d'entraîner un réseau de neurones sur graphes à en extraire les données contextualisées utiles pour prédire le succès et la qualité d'une insertion dans des tournées optimisées.

8.3 Traitement des données

Si l'optimisation nécessite un travail en amont qu'est la conception d'un programme mathématique compatible avec les données fournies et exploitable par un solveur, le succès de l'application de l'apprentissage automatique dépend crucialement des données disponibles et de leur qualité en apprentissage supervisé, ainsi que de la conception de l'exploration des possibilités en apprentissage non-supervisé ou par renforcement. Chapitre 5 décrit l'usage de données historiques de requêtes de transport médical. Si le choix de l'apprentissage par renforcement multi-agent était initialement envisagé, celui-ci s'est révélé peu stable et très dépendant de la qualité et la performance du simulateur utilisé pour l'entraînement. L'apprentissage supervisé pour la prévision par zone des futures requêtes s'est révélée plus efficace et sujette à moins de choix de conception ou d'augmentation de données, bien que proposant

une approche plus limitée. La stabilité et simplicité de cette approche fut un point positif dans la collaboration avec les répartiteurs à qui l'outil était destiné. Dans le cas de l'insertion de rendez-vous dans des tournées, les données disponibles sous forme d'emplois du temps journaliers étaient produites par des solveurs mathématiques dont les résultats furent modifiés à la main par un répartiteur humain. L'intervention humaine sur les données étant problématique dans une étude visant à anticiper les résultats d'une optimisation mathématique, les emplois du temps furent tous recréés par un nouveau solveur permettant de créer des résolutions cohérentes, et donc un ensemble d'entraînement cohérent. Cette approche fut longue en temps de calcul et en échange avec les décideurs concernés par ces emplois du temps, comme il convient de s'assurer de leur pertinence, mais permit de poser un cadre précis à l'utilisation des données par le prédicteur. Dans le cadre de problèmes combinatoires définis sur des graphes, comme pour le stable maximum ou le voyageur de commerce des Chapitres 4 et 7, les performances dépendent de la distance du graphe considéré à la distribution vue durant l'entraînement ; il convient donc d'entraîner ses agents sur des distributions de graphe adéquates et représentatives de problèmes rencontrés. L'apprentissage non-supervisé et par renforcement ne dépendent pas de réponses de référence fournies avec les exemples d'entraînement, mais d'un score, une valeur quantifiant la qualité des décisions prises ou des valeurs générés. La conception de ce score, ou fonction de récompense, est alors cruciale à l'apprentissage puisqu'elle en sera le guide.

CHAPITRE 9 CONCLUSION ET RECOMMANDATIONS

Les projets présentés dans cette thèse démontrent le potentiel de l'application de techniques d'apprentissage automatique, et particulièrement d'apprentissage profond, en recherche opérationnelle. Dans ce chapitre nous revenons sur les contributions présentées et discutons leurs limitations.

9.1 Synthèse des travaux

L'approche la plus simple et la plus classique en apprentissage profond, l'apprentissage supervisé, permet d'apprendre rapidement et avec relativement peu de données une estimation des requêtes futures pour le repositionnement d'ambulances, Chapitre 5. Cette approche, dite « estimation puis optimisation » permet d'estimer des paramètres inconnus à la lumière de données historiques afin de prendre des décisions plus éclairées ; dans le cas des ambulances, on estime ainsi quelles sont les zones les plus susceptibles de nécessiter une ambulance en fonction de l'activité passée. Chapitre 6 présente une autre application de l'apprentissage supervisé, cette fois-ci associé à une représentation graphique du problème : un problème d'optimisation et une solution sont représentés par un graphe riche en informations mettant en relation les divers acteurs du problème afin d'en extraire une représentation riche grâce à un réseau de neurone sur graphes.

Le Chapitre 4 quant à lui présente une approche plus sophistiquée avec l'utilisation d'apprentissage par renforcement pour d'explorer les possibilités offerte par l'ordonnancement des couches de diagrammes de décision et leur impact sur les bornes d'optimisation au sein d'un algorithme de branch-and-bound. L'apprentissage par renforcement permet ainsi la découverte d'heuristiques, complexes et peu interprétables, mais adaptées aux problèmes traités et sans intervention humaine. L'exploration autonome des états possibles par actions successives en quantifiant la qualité des diagrammes de décision construits permet de ne pas limiter l'agent d'apprentissage par des heuristiques ou solutions de référence limitées puisque l'ordonnancement optimal des couches n'est pas connu. L'impact de l'agent sur la recherche par branch-and-bound est certain, notamment sur la réduction de la taille de l'arbre de recherche, mais une diminution du temps de résolution est plus ardue car il faut que le gain en réduction de l'arbre de recherche compense le coût d'appels successifs à l'agent. L'approche proposée permet finalement d'obtenir des gains de 20% sur le nombre de noeuds explorés et de 15% sur le temps de résolution en limitant le nombre d'appels à l'agent durant la résolution.

Finalement, une approche d'apprentissage non-supervisé, présentée Chapitre 7, permet de générer des facteurs lagrangiens pour la relaxation de Held et Karp du voyageur de commerce. La qualité de ces facteurs est évaluée par la borne qu'ils permettent de produire. Ce processus de génération de borne est inversé pour ajuster la génération des facteurs lagrangiens pour la maximisation de cette borne. Ainsi l'entraînement permet la génération de facteurs de plus en plus performants, sans connaître ni avoir besoin de connaître des facteurs de référence, et une application à un algorithme de branch-and-bound pour la résolution du voyageur de commerce résulte en des résolutions 20% plus rapides sur les instances testées.

9.2 Limitations et améliorations futures

Dans le premier article, présenté Chapitre 4, une heuristique de construction automatique découverte par apprentissage par renforcement est intégrée à un algorithme de branch-and-bound ; la qualité de cette heuristique est directement dépendante du succès et de la stabilité de l'entraînement de l'agent et pourrait être améliorée en dérivant des règles mathématiques qui en décrivent totalement la logique. Ce travail est un champ de recherche développement et n'est pas garanti de fournir un ensemble concis et satisfaisant de règles menant à des performances similaires à celles d'un agent entraîné, mais pourrait grandement améliorer l'interprétabilité des décisions prises, pourrait fournir des idées précieuses pour l'amélioration des phases d'exploration des solveurs pour le problème en question ou dans un contexte plus général, et permettrait potentiellement la simplification de la mise en place de l'heuristique en s'affranchissant des inférences de réseaux de neurones. Dans une moindre mesure, des techniques de distillation de connaissance (*knowledge distillation*) pourraient permettre d'approximer l'heuristique sur de plus petits modèles. Un autre axe d'étude à considérer pour les applications pratiques concerne la scalabilité des différents modèles développés à des tailles de problèmes différentes. Si l'utilisation de réseaux de neurones sur graphes présente pour avantage de ne pas limiter la taille du graphe en entrée du modèle, un entraînement adapté aux tailles de problèmes rencontrées est tout de même nécessaire pour obtenir les meilleures performances. Le transfert d'apprentissage pourrait ainsi permettre d'effectuer de courts entraînements d'un modèle pré-entraîné sur une nouvelle distribution, afin d'en adapter le comportement sans reprendre de zéro un entraînement coûteux. Cette approche est un champ de recherche actif qui permettrait l'adaptation rapide des modèles développés à de nouvelles tailles de problèmes, de nouvelles instances, voire de nouvelles tâches en fonction des performances qu'il serait possible d'obtenir.

Les projets présentés Chapitres 5 et 6 sont destinés à des applications concrètes présentant des limitations et exigences plus pratiques. La génération de données pour l'insertion de

rendez-vous nécessite la résolution de nombreux programmes de VRP afin d'en fournir les solutions à un prédicteur essayant de les approximer. Ce genre d'approche illustre les limites de l'apprentissage par renforcement dans les cas où la génération ou l'obtention de références pour l'apprentissage est coûteuse. Dans le cas général, cela fait de l'apprentissage supervisé une méthode coûteuse si elle a pour objectif d'imiter le résultat, ou une partie de résultat, d'un programme d'optimisation. Les outils produits lors de ces projets étant destinés à être utilisés par des répartiteurs, une source d'amélioration de l'expérience utilisateur serait la prise en compte de leurs préférences dans les comportements des prédicteurs. Si celles-ci sont partiellement incorporées lors de la phase de conception menée conjointement avec des répartiteurs, toutes les possibilités ne peuvent pas être communiquées ou envisagées. Une prise en compte des retours utilisateurs a posteriori, par exemple par un ré-entraînement sur les exemples incorporant le retour utilisateur, pourrait apporter une plus-value non négligeable, au coût du développement d'un système de collecte de ces avis et d'une stratégie d'incorporation. Un point de critique intéressant concernant ces approches dites d'« estimation puis optimisation » concerne le fait qu'une petite imprécision sur une prédiction peut mener à une grande erreur sur le résultat d'une optimisation. Des approches plus sophistiquées intégrant directement la décision finale dans le processus d'apprentissage sont alors d'intérêt et présentent une direction de recherche prometteuse pour l'intégration de l'apprentissage et de la recherche opérationnelle. L'apprentissage tirant largement parti de la dérivabilité des différentes étapes en jeux, l'intégration d'une étape de prise de décision, par exemple par la résolution d'un programme linéaire en nombres entiers, est un sujet de recherche actif et mène au développement de nouvelles approches intégrées, dites d'« estimation et optimisation ».

En conclusion, de nombreuses directions de recherche restent à explorer dans l'intégration de l'apprentissage automatique et de la recherche opérationnelle, du fait des avancées indépendantes de ces deux domaines aux interconnexions nombreuses. Des développements technologiques plus poussés et des efforts de réflexion plus unifiés sont les fers de lance de la recherche actuelle, et ils se basent sur des avancées théoriques et techniques rapides et récentes à l'échelle des sciences.

RÉFÉRENCES

- [1] G. B. Dantzig, “Inductive proof of the simplex method,” *IBM Journal of Research and Development*, vol. 4, n^o. 5, p. 505–506, 1960.
- [2] ———, “Origins of the simplex method,” dans *A History of Scientific Computing*. New York, NY, USA : Association for Computing Machinery, 1990, p. 141–151. [En ligne]. Disponible : <https://doi.org/10.1145/87252.88081>
- [3] N. Karmarkar, “A new polynomial-time algorithm for linear programming,” *Combinatorica*, vol. 4, p. 373–395, 1984.
- [4] A. H. Land et A. G. Doig, “An automatic method of solving discrete programming problems,” *Econometrica*, vol. 28, n^o. 3, p. 497–520, 1960. [En ligne]. Disponible : <http://www.jstor.org/stable/1910129>
- [5] J. D. C. Little, K. G. Murty, D. W. Sweeney et C. Karel, “An algorithm for the traveling salesman problem,” *Operations Research*, vol. 11, n^o. 6, p. 972–989, 1963. [En ligne]. Disponible : <https://doi.org/10.1287/opre.11.6.972>
- [6] M. et G. Rinaldi, “A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems,” *SIAM Review*, vol. 33, n^o. 1, p. 60–100, 1991. [En ligne]. Disponible : <https://doi.org/10.1137/1033004>
- [7] G. E. Moore, “Progress in digital integrated electronics [technical literature, copyright 1975 ieee. reprinted, with permission. technical digest. international electron devices meeting, ieee, 1975, pp. 11-13.],” *IEEE Solid-State Circuits Society Newsletter*, vol. 11, n^o. 3, p. 36–37, 2006.
- [8] M. Waldrop, “The chips are down for moore’s law,” *Nature*, vol. 530, p. 144–147, 2016.
- [9] J. Benders, “Partitioning procedures for solving mixed-variables programming problems,” *Numerische Mathematik*, vol. 4, p. 238–252, 1962.
- [10] G. D. P. Wolfe, “Decomposition principle for linear programs,” *Operations Research*, vol. 8, p. 101–111, 1960.
- [11] G. Desaulniers, J. Desrosiers et M. Solomon, *Column Generation*. Springer-Verlag New York Inc, 2005.
- [12] G. Gutin et A. P. Punnen, *The traveling salesman problem and its variations*. Springer New York, NY, 2007.

- [13] D. L. Applegate, R. E. Bixby, V. Chvátal et W. J. Cook, *The Traveling Salesman Problem : A Computational Study*. Princeton University Press, 2006. [En ligne]. Disponible : <http://www.jstor.org/stable/j.ctt7s8xg>
- [14] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023. [En ligne]. Disponible : <https://www.gurobi.com>
- [15] I. I. Cplex, “V12. 1 : User’s manual for cplex,” *International Business Machines Corporation*, vol. 46, n°. 53, p. 157, 2009.
- [16] E. L. Lawler et D. E. Wood, “Branch-and-bound methods : A survey,” *Operations Research*, vol. 14, n°. 4, p. 699–719, 1966. [En ligne]. Disponible : <http://www.jstor.org/stable/168733>
- [17] S. Ahmed, M. Tawarmalani et N. V. Sahinidis, “A finite branch-and-bound algorithm for two-stage stochastic integer programs,” *Mathematical Programming*, vol. 100, p. 355–377, 2004.
- [18] D. R. Morrison, S. H. Jacobson, J. J. Sauppe et E. C. Sewell, “Branch-and-bound algorithms : A survey of recent advances in searching, branching, and pruning,” *Discrete Optimization*, vol. 19, p. 79–102, 2016. [En ligne]. Disponible : <https://www.sciencedirect.com/science/article/pii/S1572528616000062>
- [19] L. Huang, X. Chen, W. Huo, J. Wang, F. Zhang, B. Bai et L. Shi, “Branch and bound in mixed integer linear programming problems : A survey of techniques and trends,” 2021.
- [20] C.-Y. Lee, “Representation of switching circuits by binary-decision programs,” *Bell system Technical journal*, vol. 38, n°. 4, p. 985–999, 1959.
- [21] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *Computers, IEEE Transactions on*, vol. 100, n°. 8, p. 677–691, 1986.
- [22] M. P. Castro, A. A. Cire et J. C. Beck, “Decision diagrams for discrete optimization : A survey of recent advances,” *INFORMS Journal on Computing*, vol. 34, n°. 4, p. 2271–2295, 2022. [En ligne]. Disponible : <https://doi.org/10.1287/ijoc.2022.1170>
- [23] D. Bergman, A. A. Cire, W.-J. van Hoeve et T. Yunes, “Bdd-based heuristics for binary optimization,” *Journal of Heuristics*, vol. 20, n°. 2, p. 211–234, 2014.
- [24] A. A. Cire et W.-J. van Hoeve, “Multivalued decision diagrams for sequencing problems,” *Operations Research*, vol. 61, n°. 6, p. 1411–1428, 2013. [En ligne]. Disponible : <https://doi.org/10.1287/opre.2013.1221>
- [25] B. Kell et W.-J. van Hoeve, “An mdd approach to multidimensional bin packing,” dans *Integration of AI and OR Techniques in Constraint Programming for Combinatorial*

- Optimization Problems*, C. Gomes et M. Sellmann, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013, p. 128–143.
- [26] D. de Uña, G. Gange, P. Schachte et P. J. Stuckey, “Compiling cp subproblems to mdds and d-dnnfs,” *Constraints*, vol. 24, p. 56–93, 2019.
 - [27] L. Lozano et J. C. Smith, “A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs,” *Mathematical Programming*, vol. 191, p. 381–404, 2022.
 - [28] Q. Cappart, E. Goutierre, D. Bergman et L.-M. Rousseau, “Improving optimization bounds using machine learning : Decision diagrams meet deep reinforcement learning,” dans *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, p. 1443–1451.
 - [29] D. Bergman, A. A. Cire, W.-J. van Hoeve et J. N. Hooker, “Discrete optimization with decision diagrams,” *INFORMS J. on Computing*, vol. 28, n^o. 1, p. 47–66, févr. 2016.
 - [30] V. C. Xavier Gillard, Pierre Schaus, “Ddo, a generic and efficient framework for mdd-based optimization,” IJCAI-20, 2020, available from <https://github.com/xgillard/ddo>.
 - [31] I. Rudich, Q. Cappart et L.-M. Rousseau, “Peel-and-bound : Generating stronger relaxed bounds with multivalued decision diagrams,” 2022.
 - [32] —, “Improved peel-and-bound : Methods for generating dual bounds with multivalued decision diagrams,” 2023.
 - [33] M. Behle, “On threshold bdds and the optimal variable ordering problem,” dans *Combinatorial Optimization and Applications*, A. Dress, Y. Xu et B. Zhu, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 124–135.
 - [34] K. Hornik, M. Stinchcombe et H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, n^o. 5, p. 359–366, 1989. [En ligne]. Disponible : <https://www.sciencedirect.com/science/article/pii/0893608089900208>
 - [35] A. Krizhevsky, I. Sutskever et G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” dans *Advances in neural information processing systems*, 2012, p. 1097–1105.
 - [36] K. Simonyan et A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.
 - [37] Y. Goldberg et O. Levy, “word2vec explained : deriving mikolov et al.’s negative-sampling word-embedding method,” 2014.
 - [38] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser,

- S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes et J. Dean, “Google’s neural machine translation system : Bridging the gap between human and machine translation,” 2016.
- [39] I. Wallach, M. Dzamba et A. Heifets, “Atomnet : A deep convolutional neural network for bioactivity prediction in structure-based drug discovery,” 2015.
- [40] A. Zhavoronkov, Y. A. Ivanenkov, A. Aliper, M. S. Veselov, V. A. Aladinskiy, A. V. Aladinskaya, V. A. Terentiev, D. A. Polykovskiy, M. D. Kuznetsov, A. Asadulaev, Y. Volkov, A. Zholus, R. R. Shayakhmetov, A. Zhebrak, L. I. Minaeva, B. A. Zagribelnyy, L. H. Lee, R. Soll, D. Madge, L. Xing, T. Guo et A. Aspuru-Guzik, “Deep learning enables rapid identification of potent ddr1 kinase inhibitors,” *Nature biotechnology*, vol. 37, n^o. 9, p. 1038—1040, September 2019. [En ligne]. Disponible : <https://doi.org/10.1038/s41587-019-0224-x>
- [41] I. Goodfellow, Y. Bengio et A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [42] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin et M. J. Cardoso, “Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations,” dans *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer International Publishing, 2017, p. 240–248. [En ligne]. Disponible : https://doi.org/10.1007%2F978-3-319-67558-9_28
- [43] M. Arjovsky, S. Chintala et L. Bottou, “Wasserstein gan,” 2017.
- [44] Q. Wang, Y. Ma, K. Zhao et Y. Tian, “A comprehensive survey of loss functions in machine learning,” *Annals of Data Science*, vol. 9, 04 2022.
- [45] D. P. Kingma et J. Ba, “Adam : A method for stochastic optimization,” *arXiv preprint arXiv :1412.6980*, 2014.
- [46] C. J. Watkins et P. Dayan, “Q-learning,” *Machine learning*, vol. 8, n^o. 3-4, p. 279–292, 1992.
- [47] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra et M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv :1312.5602*, 2013.
- [48] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, n^o. 7540, p. 529, 2015.
- [49] J. Schulman, F. Wolski, P. Dhariwal, A. Radford et O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv :1707.06347*, 2017.

- [50] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver et K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” dans *International conference on machine learning*, 2016, p. 1928–1937.
- [51] Y. Lecun, L. Bottou, Y. Bengio et P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, n°. 11, p. 2278–2324, 1998.
- [52] S. Hochreiter et J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, n°. 8, p. 1735–1780, 11 1997. [En ligne]. Disponible : <https://doi.org/10.1162/neco.1997.9.8.1735>
- [53] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville et Y. Bengio, “Generative adversarial networks,” 2014.
- [54] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner et G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks*, vol. 20, n°. 1, p. 61–80, 2009.
- [55] T. N. Kipf et M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv :1609.02907*, 2016.
- [56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser et I. Polosukhin, “Attention is all you need,” 2017.
- [57] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris et P. Veličković, “Combinatorial optimization and reasoning with graph neural networks,” *arXiv preprint arXiv :2102.09544*, 2021.
- [58] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. M. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter et J. Dean, “A graph placement methodology for fast chip design,” *Nature*, vol. 594, p. 207 – 212, 2021.
- [59] D. Bahdanau, K. Cho et Y. Bengio, “Neural machine translation by jointly learning to align and translate,” 2016.
- [60] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò et Y. Bengio, “Graph Attention Networks,” *International Conference on Learning Representations*, 2018. [En ligne]. Disponible : <https://openreview.net/forum?id=rJXMpikCZ>
- [61] J. Hopfield et D. Tank, ““neural” computation of decisions in optimization problems,” *Biological Cybernetics*, vol. 52, p. 141–152, 1985.
- [62] A. Vakhutinsky et B. Golden, “Solving vehicle routing problems using elastic nets,” dans *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN’94)*, vol. 7, 1994, p. 4535–4540 vol.7.

- [63] —, “A hierarchical strategy for solving traveling salesman problems using elastic nets,” *Journal of Heuristics*, vol. 1, p. 67–76, 1995.
- [64] K. A. Smith, “Neural networks for combinatorial optimization : A review of more than a decade of research,” *INFORMS Journal on Computing*, vol. 11, n^o. 1, p. 15–34, 1999. [En ligne]. Disponible : <https://doi.org/10.1287/ijoc.11.1.15>
- [65] C.-K. Looi, “Neural network methods in combinatorial optimization,” *Computers and Operations Research*, vol. 19, n^o. 3, p. 191–208, 1992. [En ligne]. Disponible : <https://www.sciencedirect.com/science/article/pii/0305054892900446>
- [66] E. Khalil, H. Dai, Y. Zhang, B. Dilkina et L. Song, “Learning combinatorial optimization algorithms over graphs,” dans *Advances in Neural Information Processing Systems*, 2017, p. 6351–6361.
- [67] F. Hutter, H. H. Hoos et K. Leyton-Brown, “Automated configuration of mixed integer programming solvers,” dans *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, A. Lodi, M. Milano et P. Toth, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, p. 186–202.
- [68] K. Yilmaz et N. Yorke-Smith, “A study of learning search approximation in mixed integer branch and bound : Node selection in SCIP,” *AI*, vol. 2, n^o. 2, p. 150–178, apr 2021.
- [69] M. Gasse, D. Chételat, N. Ferroni, L. Charlin et A. Lodi, “Exact combinatorial optimization with graph convolutional neural networks,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [70] A. Lodi et G. Zarpellon, “On learning and branching : a survey,” *TOP*, vol. 25, n^o. 2, p. 207–236, 2017.
- [71] I. Bello, H. Pham, Q. V. Le, M. Norouzi et S. Bengio, “Neural combinatorial optimization with reinforcement learning,” 2017.
- [72] A. N. Elmachoub et P. Grigas, “Smart "predict, then optimize",” 2020.
- [73] A. Ferber, B. Wilder, B. Dilkina et M. Tambe, “Mipaal : Mixed integer program as a layer,” 2019.
- [74] A. Paleyes, R.-G. Urma et N. D. Lawrence, “Challenges in deploying machine learning : A survey of case studies,” *ACM Computing Surveys*, vol. 55, n^o. 6, p. 1–29, dec 2022. [En ligne]. Disponible : <https://doi.org/10.1145%2F3533378>
- [75] T. Hadzic et J. Hooker, “Postoptimality analysis for integer programming using binary decision diagrams,” dans *GICOLAG Workshop (Global Optimization)*, Vienna. Technical report, Carnegie Mellon University, 2006.

- [76] H. R. Andersen, T. Hadzic, J. N. Hooker et P. Tiedemann, “A constraint store based on multivalued decision diagrams,” dans *International Conference on Principles and Practice of Constraint Programming*. Springer, 2007, p. 118–132.
- [77] D. Bergman, A. A. Cire, W.-J. van Hove et J. Hooker, *Decision diagrams for optimization*. Springer, 2016.
- [78] D. Bergman, W.-J. van Hove et J. N. Hooker, “Manipulating MDD relaxations for combinatorial optimization,” dans *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, p. 20–35.
- [79] Y. LeCun, Y. Bengio et G. Hinton, “Deep learning,” *Nature*, vol. 521, p. 436–44, 05 2015.
- [80] K. Arulkumaran, M. P. Deisenroth, M. Brundage et A. A. Bharath, “A brief survey of deep reinforcement learning,” *CoRR*, vol. abs/1708.05866, 2017. [En ligne]. Disponible : <http://arxiv.org/abs/1708.05866>
- [81] R. S. Sutton et A. G. Barto, *Reinforcement learning : An introduction*. MIT press, 2018.
- [82] H. Dai, B. Dai et L. Song, “Discriminative embeddings of latent variable models for structured data,” dans *International Conference on Machine Learning*, 2016, p. 2702–2711.
- [83] D. Bergman, A. A. Cire, W.-J. van Hove et J. N. Hooker, “Variable ordering for the application of BDDs to the maximum independent set problem,” dans *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 2012, p. 34–49.
- [84] R. Albert et A.-L. Barabási, “Statistical mechanics of complex networks,” *Reviews of Modern Physics*, vol. 74, p. 47–97, janv. 2002.
- [85] J. J. Moré et E. D. Dolan, “Benchmarking optimization software with performance profiles,” *Mathematical Programming*, vol. 91, 2002.
- [86] P. Gupta, M. Gasse, E. Khalil, P. Mudigonda, A. Lodi et Y. Bengio, “Hybrid models for learning to branch,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.
- [87] R. J. O’Neil et K. Hoffman, “Decision diagrams for solving traveling salesman problems with pickup and delivery in real time,” *Operations Research Letters*, vol. 47, n^o. 3, p. 197 – 201, 2019.

- [88] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak et L.-M. Rousseau, “Learning heuristics for the tsp by policy gradient,” dans *International conference on the integration of constraint programming, artificial intelligence, and operations research*. Springer, 2018, p. 170–181.
- [89] M. Gendreau, G. Laporte et F. Semet, “Solving an ambulance location model by tabu search,” *Location Science*, vol. 5, n^o. 2, p. 75–88, 1997.
- [90] J. B. Goldberg, “Operations research models for the deployment of emergency services vehicles,” *EMS Management Journal*, vol. 1, n^o. 1, p. 20–39, 2004.
- [91] M. Held et R. M. Karp, “A dynamic programming approach to sequencing problems,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, n^o. 1, p. 196–210, 1962.
- [92] G. J. Woeginger, *Exact Algorithms for NP-Hard Problems : A Survey*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2003, p. 185–207.
- [93] S. Kumar et R. Panneerselvam, “A survey on the vehicle routing problem and its variants,” *Intelligent Information Management*, vol. 4, n^o. 3, p. 66–74, 2012.
- [94] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera et A. A. Juan, “Rich vehicle routing problem : Survey,” *ACM Comput. Surv.*, vol. 47, n^o. 2, 2014.
- [95] C. Lin, K. Choy, G. Ho, S. Chung et H. Lam, “Survey of green vehicle routing problem : Past and future trends,” *Expert Systems with Applications*, vol. 41, n^o. 4, Part 1, p. 1118–1138, 2014.
- [96] E. C. Jones et B. D. Leibowicz, “Contributions of shared autonomous vehicles to climate change mitigation,” *Transportation Research Part D : Transport and Environment*, vol. 72, p. 279–298, 2019.
- [97] D. Bertsimas, P. Jaillet et S. Martin, “Flexbus : Improving public transit with ride-hailing technology,” *Dow Sustainability Fellowship*, 2017. [En ligne]. Disponible : <http://sustainability.umich.edu/media/files/dow/Dow-Masters-Report-FlexBus.pdf>
- [98] H. N. Psaraftis, M. Wen et C. A. Kontovas, “Dynamic vehicle routing problems : Three decades and counting,” *Networks*, vol. 67, n^o. 1, p. 3–31, 2016.
- [99] S. C. Ho, W. Szeto, Y.-H. Kuo, J. M. Leung, M. Petering et T. W. Tou, “A survey of dial-a-ride problems : Literature review and recent developments,” *Transportation Research Part B : Methodological*, vol. 111, p. 395–421, 2018.
- [100] D.-H. Lee, H. Wang, R. Cheu, H. Siew et Teo, “A taxi dispatch system based on current demands and real-time traffic conditions,” *Transportation Research Record*, vol. 1882, 01 2004.

- [101] J. Bischoff et M. Maciejewski, “Simulation of city-wide replacement of private cars with autonomous taxis in berlin,” *The 7th International Conference on Ambient Systems, Networks and Technologies*, vol. 83, p. 237–244, 2016.
- [102] K. T. Seow, N. H. Dang et D.-H. Lee, “A collaborative multiagent taxi-dispatch system,” *IEEE Transactions on Automation Science and Engineering*, vol. 7, n°. 3, p. 607–616, 2010.
- [103] F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He et G. J. Pappas, “Taxi dispatch with real-time sensing data in metropolitan areas : A receding horizon control approach,” *IEEE Transactions on Automation Science and Engineering*, vol. 13, n°. 2, p. 463–478, 2016.
- [104] R. Zhang et M. Pavone, “Control of robotic mobility-on-demand systems : A queueing-theoretical perspective,” *The International Journal of Robotics Research*, vol. 35, n°. 1-3, p. 186–203, 2016.
- [105] A. Braverman, J. G. Dai, X. Liu et L. Ying, “Empty-car routing in ridesharing systems,” *Operations Research*, vol. 67, n°. 5, p. 1437–1452, 2019.
- [106] D. J. Fagnant et K. M. Kockelman, “The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios,” *Transportation Research Part C : Emerging Technologies*, vol. 40, p. 1–13, 2014.
- [107] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli et D. Rus, “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment,” *Proceedings of the National Academy of Sciences*, vol. 114, n°. 3, p. 462–467, 2017.
- [108] F. Dandl, M. Hyland, K. Bogenberger et H. S. Mahmassani, “Evaluating the impact of spatio-temporal demand forecast aggregation on the operational performance of shared autonomous mobility fleets,” *Transportation*, vol. 46, p. 1975–1996, 2019.
- [109] D. Bertsimas, P. Jaillet et S. Martin, “Online vehicle routing : The edge of optimization in large-scale applications,” *Operations Research*, vol. 67, n°. 1, p. 143–162, 2019.
- [110] F. Rossi, R. Zhang et M. Pavone, “Routing autonomous vehicles in congested transportation networks : structural properties and coordination algorithms,” *Autonomous Robots*, vol. 42, p. 1427–1442, 2018.
- [111] M. Gmira, M. Gendreau, A. Lodi et J.-Y. Potvin, “Managing in real-time a vehicle routing plan with time-dependent travel times on a road network,” *Transportation Research Part C : Emerging Technologies*, vol. 132, p. 103379, 2021.
- [112] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,

- M. Brucher, M. Perrot et E. Duchesnay, “Scikit-learn : Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, p. 2825–2830, 2011.
- [113] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus et Y. LeCun, “Overfeat : Integrated recognition, localization and detection using convolutional networks,” 2014.
- [114] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai et X. Huang, “Pre-trained models for natural language processing : A survey,” *Science China Technological Sciences*, vol. 63, n^o. 10, p. 1872–1897, 2020.
- [115] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine *et al.*, “Model-based reinforcement learning for atari,” *International Conference on Learning Representations, ICLR*, 2020.
- [116] Y. Bengio, A. Lodi et A. Prouvost, “Machine learning for combinatorial optimization : A methodological tour d’horizon,” *European Journal of Operational Research*, vol. 290, n^o. 2, p. 405–421, 2021.
- [117] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian et J. Ye, “Large-scale order dispatch in on-demand ride-hailing platforms : A learning and planning approach,” *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, p. 905 – 913, 2018.
- [118] T. Oda et C. Joe-Wong, “Movi : A model-free approach to dynamic fleet management,” dans *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. IEEE Press, 2018, p. 2708–2716.
- [119] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang et J. Ye, “Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem,” dans *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE Computer Society, 2019, p. 1090–1095.
- [120] N. D. Kullman, M. Cousineau, J. C. Goodson et J. E. Mendoza, “Dynamic ride-hailing with electric vehicles,” *Transportation Science*, vol. 56, n^o. 3, p. 775–794, 2022.
- [121] C. Riley, P. van Hentenryck et E. Yuan, “Real-time dispatching of large-scale ride-sharing systems : Integrating optimization, machine learning, and model predictive control,” dans *IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, 7 2020, p. 4417–4423, special track on AI for CompSust and Human well-being.
- [122] E. Yuan, W. Chen et P. Van Hentenryck, “Reinforcement learning from optimization proxy for ride-hailing vehicle relocation,” *Journal of Artificial Intelligence Research (JAIR)*, vol. 75, dec 2022.

- [123] Y. Jiao, X. Tang, Z. T. Qin, S. Li, F. Zhang, H. Zhu et J. Ye, “Real-world ride-hailing vehicle repositioning using deep reinforcement learning,” *Transportation Research Part C : Emerging Technologies*, vol. 130, p. 103289, 2021.
- [124] Q. Luo et X. Huang, “Multi-agent reinforcement learning for empty container repositioning,” dans *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, 2018, p. 337–341.
- [125] K. Liu, X. Li, C. C. Zou, H. Huang et Y. Fu, “Ambulance dispatch via deep reinforcement learning,” ser. SIGSPATIAL ’20. Association for Computing Machinery, 2020, p. 123–126.
- [126] D. P. Kingma et J. Ba, “Adam : A method for stochastic optimization,” *arXiv preprint arXiv :1412.6980*, 2014.
- [127] G. Laporte, P. Toth et D. Vigo, “Vehicle routing : historical perspective and recent contributions,” *EURO Journal on Transportation and Logistics*, vol. 2, p. 1–4, 2013.
- [128] H. N. Psaraftis, M. Wen et C. A. Kontovas, “Dynamic vehicle routing problems : Three decades and counting,” *Networks*, vol. 67, n^o. 1, p. 3–31, 2016. [En ligne]. Disponible : <https://onlinelibrary.wiley.com/doi/abs/10.1002/net.21628>
- [129] M. Gendreau, G. Laporte et R. Séguin, “Stochastic vehicle routing,” *European Journal of Operational Research*, vol. 88, n^o. 1, p. 3–12, 1996. [En ligne]. Disponible : <https://www.sciencedirect.com/science/article/pii/037722179500050X>
- [130] A. Mor et M. G. Speranza, “Vehicle routing problems over time : a survey,” *A Quarterly Journal of Operations Research (4OR)*, vol. 18, p. 129–149, 2020.
- [131] A. Bogrybayeva, M. Meraliyev, T. Mustakhov et B. Dauletbayev, “Learning to solve vehicle routing problems : A survey,” 2022.
- [132] J. Fitzpatrick, D. Ajwani et P. Carroll, “Learning to sparsify travelling salesman problem instances,” 2021.
- [133] P. R. de O. da Costa, J. Rhugenaath, Y. Zhang et A. Akcay, “Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning,” 2020.
- [134] L. Perron et V. Furnon, “Or-tools,” Google. [En ligne]. Disponible : <https://developers.google.com/optimization/>
- [135] M. Held et R. M. Karp, “The traveling-salesman problem and minimum spanning trees,” *Operations Research*, vol. 18, n^o. 6, p. 1138–1162, 1970. [En ligne]. Disponible : <http://www.jstor.org/stable/169411>
- [136] —, “The traveling-salesman problem and minimum spanning trees : Part ii,” *Mathematical Programming*, vol. 18, n^o. 1, p. 6–25, 1971. [En ligne]. Disponible : <https://doi.org/10.1007/BF01584070>

- [137] Y. Bengio, A. Lodi et A. Prouvost, “Machine learning for combinatorial optimization : a methodological tour d’horizon,” 2020.
- [138] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li et Z. Zhang, “Deep graph library : A graph-centric, highly-performant package for graph neural networks,” *arXiv preprint arXiv :1909.01315*, 2019.
- [139] A. Parjadis, Q. Cappart, L.-M. Rousseau et D. Bergman, “Improving branch-and-bound using decision diagrams and reinforcement learning,” dans *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2021, p. 446–455.
- [140] M. Grötschel, L. Lovasz et A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*. Springer, 1993.
- [141] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, 2001.
- [142] T. Schaul, J. Quan, I. Antonoglou et D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv :1511.05952*, 2015.
- [143] C. L. D. Masters, “Revisiting small batch training for deep neural networks,” *arXiv :1804.07612*, 2018.
- [144] R. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, 1988.

ANNEXE A SUPPLÉMENTS DU CHAPITRE 4

A.1 Extract : Improving Variable Orderings of Approximate Decision Diagrams using Reinforcement Learning

Quentin Cappart^{ab}, Louis-Martin Rousseau^{ab}, David Bergman^c, Isabeau Prémont-Schwarz^d, Augustin Parjadis^{ab}

^aDepartment of Mathematics and Industrial Engineering, Polytechnique Montréal,

^bCIRRELT, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, ^cUniversity of Connecticut, USA, ^dServiceNow Research (formerly Element AI), Montréal, Canada

Cet article a été soumis à *INFORMS Journal on Computing* le 4 février 2022 et a été publié le 4 mai 2022. Augustin Parjadis n'est pas auteur principal de l'article, mais a aidé à sa rédaction et a rédigé intégralement la section présentée.

ABSTRACT

Prescriptive analytics provides organizations with scalable solutions for large-scale, automated decision-making. At the core of prescriptive analytics methodology is optimization, a field devoted to the study of algorithms that solve complex decision-making problems. Optimization algorithms rely heavily on generic methods for identifying tight bounds, which provide both solutions to problems and optimality guarantees. In the last decade, decision diagrams (DDs) have demonstrated significant advantages in obtaining bounds compared with the standard linear relaxation commonly used by commercial solvers. However, it is well-known that the quality of the bounds achieved by DDs is reliant on the ordering of variables chosen for the construction. Finding an ordering that optimizes standard metrics is an NP-hard problem. This paper studies how machine learning, specifically deep reinforcement learning (DRL), can be used to improve bounds provided by DDs, in particular through learning a good variable ordering. The introduced DRL models improve primal and dual bounds, even over standard linear programming relaxations, and are integrated in a full-fledged branch-and-bound algorithm. This paper therefore provides a novel mechanism for utilizing machine learning to tighten bounds, adding to recent research on using machine learning to obtain high-quality heuristic solutions and, for the first time, using machine learning to improve relaxation bounds through a generic bounding method. We apply the methods on a classic

optimization problem, the maximum independent set, and demonstrate through computational testing that optimization bounds can be significantly improved through DRL. We provide the code to replicate the results obtained on the maximum independent set.

5.3.6. Application to a Branch-and-Bound Algorithm

Following [139], we integrate the learned bounds in a DD-based branch-and-bound algorithm [29] in order to assess the practical use of the framework. Compared to the work in [139], we conduct experiments on graphs with up to 300 nodes (instead of 120), and the caching mechanism has been dropped in favor of other improvements in the efficient RL agent integration. This algorithm works by building relaxed and restricted decision diagrams at each node of the branch-and-bound search in order to obtain both primal and dual bounds, and uses relaxed DDs for branching decisions. We propose two ways to integrate our framework within this process. In the first method, the trained models are called at each node to build the decision diagrams (DD-RL). Alternatively, we provide a more efficient integration designed to obtain a trade-off between the quality of the bounds and the execution time (DD-RL+). The rationale behind this second integration is that calling a neural network many times at each node comes with significant overhead, and this provides a mechanism by which this overhead can be mitigated. The improvements of DD-RL+ include the following : (1) only the relaxed DD is built using the trained model, (2) the model is called only for the first 50 search nodes, then the MIN heuristic is used, and (3) at each prediction, the next m variables are selected instead of only the single next variable. This is done by taking m variables with the highest Q-values. These three improvements were designed empirically. We observed that the primal bound has a negligible impact on the tree-search size compared to the dual bound, and that the bounds found at the beginning of the search are the most important. The last modification enables the selection of several variables with a single call to the model. It incurs a slight loss in the bound quality in exchange for a reduction in execution time by a factor m . We set m to 5 in the reported results. Both methods are first compared to a DD-based branch-and-bound algorithm resorting to the MIN heuristic for the variable ordering. Results are presented in Figure A.1(a) for the number of nodes explored before proving optimality, and in Figure A.1(b) for the related execution time. Interestingly, we observe that even if a major reduction in the number of nodes can be obtained by DD-RL versus MIN, the execution time is dramatically worse with the learned bounds. This is due to the expensive call of the neural networks required to compute the bounds. The improvements of DD-RL+ show both a reduction in search nodes and execution time.

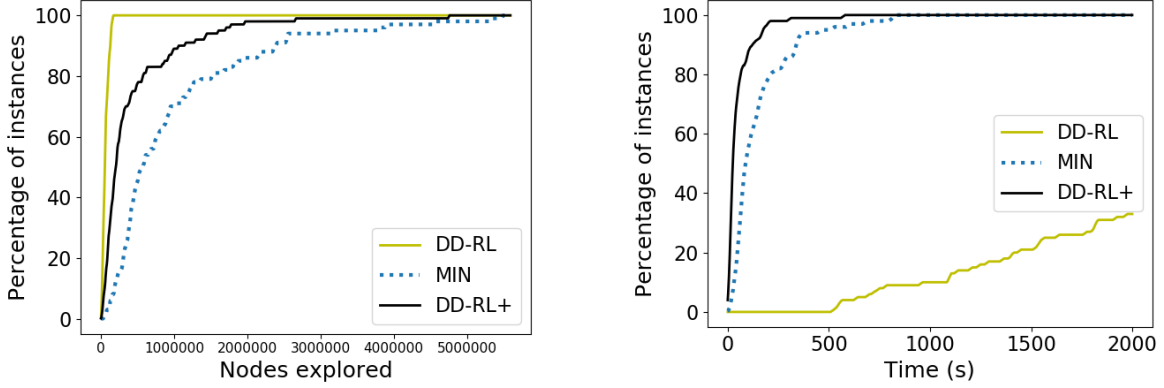


FIGURE A.1 Branch-and-bound results for 100 MISP instances (Barabasi-Albert graphs of 140 nodes, with $\nu = 4$). The y -axis represents the percentage of instances that are able to solve the problem at optimality, within a given number of nodes explored (x -axis). These results show that the our learned solver without improvement (DD-RL) is able to reduce the number of nodes explored compared to the commonly used heuristic (MIN), but at the expense of a significantly longer execution time. The improved version of the solver (DD-RL+) is able to compensate for this limitation.

5.3.7. Comparisons against CPLEX

This final experiment aims to validate our approach against an industrial and efficient MIP solver (CPLEX), based on the linear relaxation and improved by other mechanisms such as cutting planes and pre-computation of cliques [140]. The tested instances were generated randomly following a Erdos-Renyi model for graphs of $n \in \{200, 250, 300\}$ nodes and density $d = 0.3$. We made this choice as the Barabasi-Albert graphs considered so far are sparse and thus are solved efficiently with a branch-and-bound algorithm. We also increased the size of the graphs. Results are reported in Table A.1 and show the number of nodes explored, the optimality gap, and the execution time. A timeout (t.o.) of 1800 seconds is used. DD-RL uses the learned bounds until the DDs reach a depth of 50 layers, and DD-RL+ uses RL agent orderings for the first 100 search nodes. The model was trained on graphs of size 250 nodes. We consistently observe the superiority of DD-RL+ in terms of execution time and optimality gap.

TABLEAU A.1 Average number of nodes, optimality gap, and execution time for solving 100 MISP instances of size n . Results show the superiority of DD-RL+ compared to CPLEX and other DD methods for the three configurations.

Algorithms	$n = 200$			$n = 250$			$n = 300$		
	Nodes ($\times 1000$)	Opt. Gap. (%)	Time (sec)	Nodes ($\times 1000$)	Opt. Gap (%)	Time (sec)	Nodes ($\times 1000$)	Opt. Gap (%)	Time (sec)
CPLEX	8771	0	432	27059	15	t.o.	-	40	t.o.
DD-MIN	998	0	66	4947	0	416	-	7.2	t.o.
DD-RL	800	0	180	3263	0	1140	-	65	t.o.
DD-RL+	879	0	61	4018	0	362	-	4.7	t.o.

ANNEXE B ANNEXES - *REPOSITIONING FLEET VEHICLES : A LEARNING PIPELINE*

B.1 Analysis : Features Importance

In addition to the learning-based repositioning strategy, the company is also interested to know which features (time, location, motives, etc.) are the most important to explain a specific decision. To do so, we use *permutation feature importance* which measures the drop in performance when the dataset is corrupted by randomly shuffling a feature [141]. Figure B.1 shows the relative importance of different features for the neural network and the gradient boosting method. The higher the number the more important ; the features are grouped by time, PRO, PRA, and prior requests motives. We observe that *arrivals* (PRA) provide the most information for predictors, as they provide information about the locations of users that might require an additional trip in the future. We confirm these results by training with only parts of the features as shown in Table B.1. As suggested by the previous analysis, motives have a small impact on performance, raising the accuracy only by one percent or two, whereas prior requests information is essential to training accurate predictors.

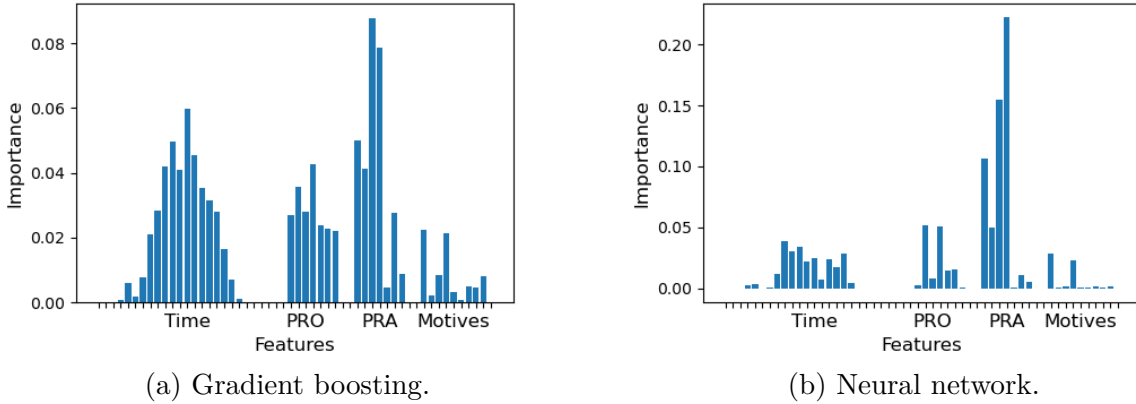


FIGURE B.1 Features importance.

B.2 Analysis : Online Learning

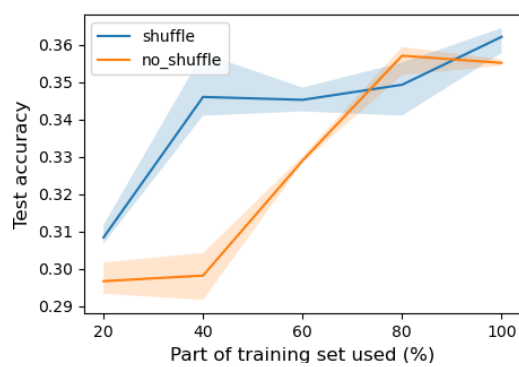
The environment in which our predictor works might vary through changes in request distribution or the way hospitals operates, so it is desirable to be able to adapt the predictor and

TABLEAU B.1 Accuracy of the different predictors for the next request.

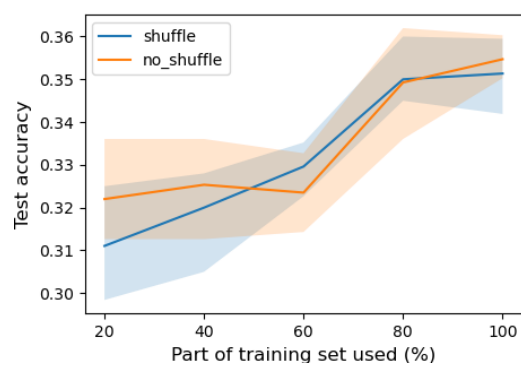
State features	NN accuracy	GradBoost accuracy
Time	28%	31%
Prior requests	32%	34%
Motives	29%	27%
Time, prior requests	33%	35%
Time, motives	30%	32%
Prior requests, motives	34%	35%
Time, prior requests, motives	35%	36%

retrain it on newer and more relevant data. This is commonly referred to as *online learning*. This can be carried out in several ways, for example, by expanding the training set with new data and retraining the model, or by freezing layers of the model and fitting on the new data. We test the efficiency of retraining approaches.

Our procedure is as follows. The dataset is split into quintiles (groups of 20% of the dataset), and at iteration $i \in \{1, \dots, 5\}$, the i first quintiles are used to train and the quintile $i + 1$ serves as the test set. This simulates new days of operation as tests, then used to retrain the predictors. This is important for further refining the accuracy of the model, and allows it to adapt to progressive environment changes ; Figure B.2 shows how accuracy evolves in time by retraining on all the data available and testing on the subsequent days of operation, averaged over 100 iterations. The quintiles are generated either by shuffling the dataset, or by keeping the data sorted time-wise to replicate retraining in practice. The temporal coherence in small data samples allows the model to capture recent temporal variations in requests but hinders access to a wider selection of examples. Overall, this appears to have a small beneficial impact on the neural network but a negative one for gradient boosting. It is clear that retraining is important for this application when there is relatively little trip data available for initial training.



(a) Gradient boosting



(b) Neural network

FIGURE B.2 Retraining accuracy

ANNEXE C SUPPLÉMENTS DU CHAPITRE 5

C.1 Ride-hailing pour une flotte de taxis

Une application naturelle du problème de ride-hailing est la gestion des taxis, pour laquelle une gestion efficace de la flotte est cruciale pour les profits totaux. Nous appliquons notre pipeline à un simulateur de taxis de la ville de New York (*pyhailing*¹) développé pour la section ride-hailing du 12ème *DIMACS Implementation Challenge* dédié aux tournées de véhicules.

C.1.1 Objectif et utilisation de la pipeline d'apprentissage

Les flottes de taxis rencontrent des défis similaires à ceux des ambulances : des demandes de courses imprévisibles doivent être prises en charge et les véhicules de la flotte doivent anticiper la répartition des demandes pour un repositionnement efficace. De plus, les taxis peuvent ici rejeter des demandes et l'objectif global est le bénéfice total, calculé comme la différence entre les gains liés aux demandes moins les coûts de fonctionnement. Le simulateur est basé sur des données des jours ouvrables de 2018 et permet de travailler sur des problèmes allant jusqu'à 500 véhicules et 50 000 demandes quotidiennes, avec 300 emplacements d'attente différents.

De manière similaire aux ambulances dans la partie 5.5.1, les données sont collectées et une représentation de l'état du problème $\mathbf{s} = (s_R, s_t)$ avec un horizon H est construite. Le problème des taxis est plus difficile en raison du plus grand nombre d'emplacements, mais de nombreuses données peuvent être obtenues pour l'entraînement. Il n'y a pas de composante de motifs de course dans l'état. L'objectif est de maximiser le bénéfice, il est donc nécessaire d'associer une bonne prédiction avec de faibles distances de déplacement pour le repositionnement (équation C).

C.1.2 Apprentissage multi-label

En raison de la grande part d'imprévisibilité dans la tâche de prédiction du prochain emplacement pour recevoir une demande, nous élargissons le problème d'apprentissage afin d'essayer de prédire les n emplacements les plus probables pour la future demande. Cela se traduit par

1. https://github.com/nkullman/ridehailing_package

un problème d'apprentissage multi-label dans lequel les n vrais labels $(y_k)_{k \in [1,n]}$ correspondent aux n emplacements auxquels des demandes sont apparues après l'état $\mathbf{s}$.

C.1.3 Heuristique et comparaisons

Nous comparons plusieurs stratégies simples de repositionnement : *aléatoire* envoie les véhicules vers n'importe quel emplacement aléatoire, *plus proche* permet aux véhicules de rester à l'emplacement le plus proche de leur arrivée de course, et *plus fréquent* envoie les véhicules vers l'emplacement le plus demandé pendant la journée non occupé par un autre véhicule.

Comme stratégie plus forte, nous présentons une heuristique de repositionnement développée par Parjadis et al. pour la piste de commande de courses du défi d'implémentation DIMACS des problèmes de routage de véhicules. Soit $\mathcal{R}$ l'ensemble des demandes précédemment reçues et V_{avail} l'ensemble des véhicules disponibles. Pour tout $r \in \mathcal{R}$ et $x \in L \cup V$, nous notons $\text{dist}[r, x]$ la distance entre la position de l'origine de la demande r et la position de l'emplacement ou du véhicule x . À chaque étape, s'il y a un véhicule v à repositionner, v est envoyé à l'emplacement l qui satisfait la condition suivante :

$$\min_{l \in L} \text{tot_dist}(l) = \sum_{d \in D} \min_{x \in V_{\text{avail}} \cup \{l\}} \text{dist}[d, x] \quad (\text{C.1})$$

avec $R \subset \mathcal{R}$ échantillonné aléatoirement avec un biais en faveur des demandes les plus récentes. Cela garantit que le véhicule sera envoyé à un endroit où il couvre des zones précédemment actives qui ne sont pas bien couvertes par les véhicules inactifs.

C.1.4 Entraînement et résultats de simulation

L'entraînement a eu lieu sur 861 000 données de trajets réparties dans un ensemble d'entraînement et un ensemble de validation (80-20%). Nous utilisons *PyTorch* pour entraîner une couche d'auto-attention couplée à un réseau de neurones à deux couches cachées de taille (300,150), et la bibliothèque *scikit-learn* pour entraîner des modèles de forêts aléatoires et de *gradient boosting*. Les paramètres ont été déterminés par recherche automatique et l'entraînement prend une heure sur un cluster d'ordinateurs universitaires équipés de processeurs Intel Gold 6148 Skylake @ 2.4 GHz.

Les prédicteurs entraînés ont une meilleure précision et variance que les autres méthodes, notamment une amélioration de 23% en termes de distance moyenne avec une variance inférieure par rapport à l'emplacement *plus fréquent*. Nous évaluons ce gain en pratique dans le tableau C.2. L'heuristique n'est pas destinée à prédire la prochaine demande, mais à déterminer où placer un véhicule afin de répartir la flotte en fonction des demandes historiques,

TABLEAU C.1 Précision des différents prédicteurs

Prédicteur	Précision	Distance moyenne à la requête observée (km)
Aléatoire	2.2%	4.63 ± 2.98
Plus près	1.9%	4.75 ± 3.16
Plus fréquent	2.2%	4.57 ± 2.43
Forêts aléatoires	2.6%	4.45 ± 2.97
NN	3.7%	3.52 ± 2.35
GradBoost	3.9%	3.55 ± 2.36

ce qui explique les performances médiocres dans le tableau C.1. L’emplacement utilisé pour calculer son score est l’emplacement de repositionnement du véhicule, et le faible score de distance moyenne indiqué s’explique par le fait que l’emplacement calculé de manière heuristique évite les emplacements où il y a déjà des véhicules, même s’il est le plus probable pour recevoir une demande. C’est un comportement souhaitable pour le score global et l’efficacité de la flotte, comme on peut le voir dans le tableau C.2.

Les prédicteurs renvoient un vecteur X qui quantifie l’importance des emplacements par rapport à la prochaine demande. Ainsi, le prochain véhicule est envoyé à l’emplacement qui minimise la distance totale entre les emplacements des véhicules inactifs, pondérée par le poids de l’emplacement dans X . Soit $F(X, \lambda)$ la fonction qui calcule cette distance totale avec un véhicule ajouté à l’emplacement λ , ce qui peut être écrit :

$$F(X, \lambda) = \sum_{l \in L} \left[X_l \min_{x \in V_L \cup \{\lambda\}} \text{dist}[l, x] \right] \quad (\text{C.2})$$

Cette fonction fournit un score de repositionnement guidé par la distribution X pour les emplacements, et permet de choisir un emplacement pour le repositionnement du véhicule en définissant

$$\text{choose_location}(X, \nu, V_L) = \text{argmin}_{l \in L} \left[F(X, l) + \text{dist}[l, \nu] \right] \quad (\text{C.3})$$

pour l’algorithme 3.

Les prédicteurs entraînés permettent de réduire considérablement la distance de réaction par rapport aux stratégies plus simples. L’*heuristique* fournit une meilleure distance de réaction mais n’atteint pas un meilleur score : cela s’explique par le fait que l’*heuristique* réagit uniquement aux demandes précédentes, tandis que les prédicteurs entraînés intègrent également une vision de la future distribution des demandes et sont donc plus prévoyants et capables de

TABLEAU C.2 Distances parcourues sur dix journées simulées

Prédicteur	Distance de réaction (de repos à requête)	Score
Plus près	6.54 ± 4.5	11590 ± 1389
Plus fréquent	5.25 ± 3.4	12685 ± 494
Aléatoire	5.39 ± 3.8	13532 ± 656
Heuristique	1.93 ± 1.6	16949 ± 304
Forêts aléatoires	1.92 ± 1.6	17180 ± 386
NN	1.94 ± 1.7	17624 ± 442
GradBoost	1.94 ± 1.7	17374 ± 324

répondre à un plus grand nombre de demandes au total. La distance de dispatching dans ces résultats est inférieure à la distance moyenne de la prédiction aux demandes, car la distance de dispatching peut être le résultat d’une attribution de véhicule autre que celui repositionné par le prédicteur.

C.1.5 Apprentissage par renforcement pour l’attribution

Jusqu’à présent, une stratégie d’attribution raisonnable consistait à envoyer le véhicule le plus proche. Cependant, plusieurs véhicules peuvent se trouver pratiquement à la même distance de l’origine de la demande, ou choisir le deuxième ou le troisième véhicule le plus proche peut être bénéfique pour la disposition globale de la flotte avec un coût minimal en temps pour le client. Nous montrons comment l’apprentissage par renforcement (RL) peut être utilisé pour améliorer la stratégie d’attribution la plus proche. Nous définissons un MDP différent avec $\mathbf{s}_t = (s_r, s_t, s_v)$ où s_r est le vecteur de demande entrante contenant l’heure d’émission et les coordonnées de l’origine et de la destination, s_t est le vecteur de temps défini comme précédemment, et s_v est le vecteur de véhicules contenant des informations sur les dix véhicules les plus proches de la demande : des booléens indiquant si le véhicule peut interrompre ses tâches pour répondre à la demande, les coordonnées absolues du véhicule, les coordonnées relatives à l’origine des demandes et sa distance par rapport à l’origine de la demande. Toutes les caractéristiques des coordonnées géographiques sont normalisées en prenant la hauteur de la carte égale à 1. Dans un souci de scalabilité, seuls les dix véhicules les plus proches disponibles sont inclus dans s_v .

L’action consiste à sélectionner l’un des 10 véhicules les plus proches disponibles ou à refuser la demande, en fonction du nœud de sortie correspondant ayant la valeur la plus élevée. Un algorithme d’apprentissage par renforcement double Q-learning est utilisé, avec une exploration ϵ -greedy. Plusieurs améliorations sont appliquées pour exploiter pleinement les données

collectées pour l'apprentissage, telles que le remplacement d'expériences priorisées [142], les mini-lots [143] et l'apprentissage en n étapes [81, 144] avec $n = 5$.

L'entraînement présenté dans la figure C.1 permet d'obtenir une stratégie d'attribution légèrement meilleure avec le réseau de neurones à couches d'attention multi-têtes (MH) par rapport à la stratégie gloutonne. Cette approche, associée à la stratégie de repositionnement étudiée précédemment, permet d'obtenir les résultats présentés dans le tableau C.3, avec les entrées en gras utilisant l'agent RL pour l'attribution ; une partition en 50 zones des 300 emplacements possibles permet de sensiblement simplifier la tâche d'apprentissage pour de meilleures performances.

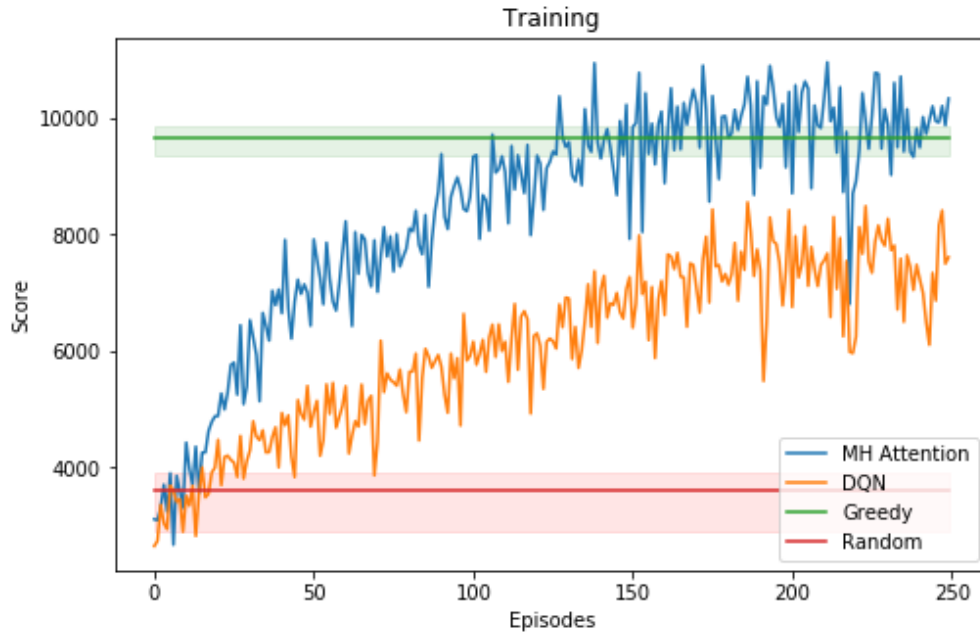


FIGURE C.1 Score d'entraînement pour un agent d'apprentissage par renforcement pour l'attribution

TABEAU C.3 Score de simulation par rapport aux participants du concours DIMACS

Repositionnement	Score (milliers)
Aléatoire	13.2
Moyenne compétition DIMACS	15.3
NN/Boost 300 emplacements	17.3
Heuristique	17.6
NN/Boost 50 zones	18.4
1ère place DIMACS	19.1

C.1.6 Discussion

Une autre méthode intéressante plus unifiée peut faire intervenir l'apprentissage par renforcement à la fois pour l'attribution et le repositionnement. Cependant, cela peut être difficile car le problème est très vaste et dynamique, mais cela a été démontré comme conduisant à de bons résultats [120]. Cette approche n'était pas adaptée pour le projet de repositionnement des ambulances décrit dans la section 5.5 en raison de la faible quantité de données disponibles. De plus, nous avons montré que des résultats satisfaisants peuvent également être obtenus pour le problème de la mise à disposition des taxis avec un pipeline d'apprentissage plus simple et plus rapide basé sur l'apprentissage supervisé, qui a été déployé et utilisé rapidement avec des partenaires industriels.

C.2 Ambulances : une approche plus précise

Le choix d'emplacement retenu pour le projet est fait par l'Algorithme 4, mais celui-ci considère uniquement les emplacements un par un jusqu'à trouver le plus prometteur. L'Équation C.3 sélectionne l'emplacement qui permet à la flotte de tendre le plus à imiter la distribution en sortie de prédicteur par sélection d'un seul emplacement, avec une pénalité de distance par rapport à l'ambulance à repositionner. Nous fournissons Table C.4 les résultats numériques de l'utilisation de l'Équation C.3 et l'Algorithme 4 comparés à *Most frequent* sur les données de la Section 5.5.4 ; si la différence est modeste pour la distance de réaction, l'Équation C.3 pour le repositionnement permet d'obtenir une distance totale plus faible au prix d'un calcul moins accessible entre terme d'interprétabilité. BI désigne les bornes inférieures qui pourraient être atteintes si toutes les requêtes pour une journée étaient connues à l'avance.

TABLEAU C.4 Distances (km) parcourues sur 10 jours simulés.

Prédicteur	Distance de réaction (de repos à requête)	Distance totale parcourue
Aléatoire	1073	1930.8
Plus prés	812.5	1412.1
Plus fréquent	587.6	1617.6
Algorithme 4		
NN	525.9 (-10%)	1552.6 (-4%)
GradBoost	526.3 (-10%)	1470.1 (-9%)
Équation C.3		
NN	519.7 (-11%)	1472.2 (-9%)
GradBoost	517.4 (-12%)	1457.5 (-10%)
BI - Distance de réaction	453.8	1721.3
BI - Distance totale	669.2	1052.1