Titre: Title:	Intégration d'apprentissage machine à un algorithme de recherche locale
Auteur: Author:	Simon Elie Bernard Charpigny
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Charpigny, S. E. B. (2024). Intégration d'apprentissage machine à un algorithme de recherche locale [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/59451/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Intégration d'apprentissage machine à un algorithme de recherche locale

SIMON ELIE BERNARD CHARPIGNY
Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie industriel

Août 2024

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Intégration d'apprentissage machine à un algorithme de recherche locale

présenté par **Simon Elie Bernard CHARPIGNY**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Thibaut VIDAL, président
Nadia LAHRICHI, membre et directrice de recherche
Daniel ALOISE, membre

DÉDICACE

A C. Dumur qui nous a laissé les meilleurs souvenirs...

REMERCIEMENTS

Je tiens tout d'abord à remercier la professeure Nadia Lahrichi pour son encadrement tout au long de ce projet. Poursuivre mon projet de recherche à ses côtés a été très formateur d'un point de vue académique et je l'en remercie.

Je remercie également tous les employés de chez AlayaCare pour m'avoir accueilli dans un environnement bienveillant. Je tiens aussi à remercier particulièrement l'équipe d'optimisation qui m'a pris en charge et dans laquelle je me suis senti tout de suite très bien intégré. Merci en particulier à François qui a consacré de son temps pour répondre à toutes mes questions et à m'apporter de ses conseils. Je garderai de très bons souvenirs de ce passage à vos côtés et de cette maîtrise.

RÉSUMÉ

Dans le domaine de l'optimisation combinatoire, l'intégration de modèles de *Reinforcement Learning* (*RL*) avec des algorithmes de recherche locale connaît un intérêt croissant. Le but recherché de cette approche est de tirer avantage des capacités d'apprentissage des modèles de *RL* et aussi de la robustesse des algorithmes de recherche locale. En effet, les méthodes de recherche locale sont connues pour rechercher des solutions de manière efficace. Cependant, elles peuvent parfois souffir d'un manque d'adaptabilité lorsque l'espace des solutions se complexifie. Cette limitation peut être surmontée par le caractère évolutif des modèles de *RL* qui modifient leur stratégie à partir de leurs expériences passées. L'apport de *RL* aux algorithmes de recherche locale permet de développer une nouvelle méthode plus robuste et capable de mieux résouder les problèmes d'optimisation complexes.

Ce mémoire explore l'intégration de techniques de (*RL*) à une métaheuristique, l'*Adaptive Large Neighborhood Search* (*ALNS*), dans le contexte de la résolution d'un problème de planification et d'ordonnancement de soins à domicile. L'étude compare les performances de *ALNS* aux algorithmes *ALNS-PPO* et *ALNS-SAC*, correspondants à *ALNS* où la partie sur la sélection des opérateurs est remplacée respectivement par les algorithmes de *RL* Proximal Policy Optimization (*PPO*) et Soft Actor-Critic (*SAC*). La comparaison des performances se fait en matière de qualité, de complexité et de vitesse de convergence des algorithmes.

Les résultats montrent que *ALNS* trouve des solutions de meilleure qualité par rapport à *ALNS-PPO* et *ALNS-SAC* sur l'ensemble des données étudiées. *ALNS* a su mieux généraliser aux variations des paramètres d'instance tout en étant plus rapide en temps de calcul. Les méthodes expérimentales incluent notamment une génération de données à partir d'un ensemble de données réelles.

Cette étude suggère plusieurs pistes de continuation, notamment la génération de données plus diversifiées et l'exploration d'autres architectures de *RL* comme les modèles en graphes ou les modèles par pointeurs (*pointer networks*). Elle suggère aussi l'exploration de nouvelles fonctions de récompense qui permettent de mieux rendre compte de la poursuite de l'objectif par le modèle.

ABSTRACT

In the field of combinatorial optimization, there is growing interest in the integration of reinforcement learning models (*RL*) with local search algorithms. The aim of this approach is to take advantage of the learning capabilities of *RL* models and the robustness of local search algorithms. Indeed, local search methods are known to search for solutions efficiently. However, they can sometimes suffer from a lack of adaptability when the solution space becomes more complex. This limitation can be overcome by the evolutionary nature of *RL* models, which modify their strategy on the basis of past experience. The contribution of *RL* to local search algorithms makes it possible to develop a new, more robust method capable of better solving complex optimization problems.

This master thesis explores the integration of (*RL*) techniques with a metaheuristic, *Adaptive Large Neighborhood Search* (*ALNS*), in the context of solving a home care planning and scheduling problem. The study compares the performance of *ALNS* with the algorithms *ALNS-PPO* and *ALNS-SAC*, corresponding to *ALNS* where the operator selection part is replaced by algorithms Proximal Policy Optimization (*PPO*) and Soft Actor-Critic (*SAC*) respectively. Performance is compared in terms of algorithm quality, complexity and convergence speed.

The results show that *ALNS* finds better quality solutions than *PPO* and *SAC* on the data set studied. *ALNS* generalizes better to variations in instance parameters, while being faster in computation time. Experimental methods included data generation from a real data set.

This study suggests several avenues for further development, including the generation of more diversified data and the exploration of other *RL* architectures such as graph models or pointer networks. It also suggests the exploration of new reward functions to better account for the model's pursuit of the goal.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ALNS	Adaptive Large Neighborhood Search
ANN	Artifical Neural Network
A3C	Asynchronous Advantage Actor Critic
B&B	Branch and Bound
COP	Combinatorial Optimization Problem
DRL	Deep Reinforcement Learning
GCN	Graph Convolutional Network
HHCRSP	Home Health Care Routing and Scheduling Problem
LNS	Large Neighborhood Search
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multilayer Perceptron
NCO	Neural Combinatorial Optimization
NP	Non-deterministic Polynomial
OP	Optimization Problem
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
TSP	Travelling Salesman Problem
VRP	Vehicle Routine Problem
VRPTW	Vehicle Routing Problem with Time Windows

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

Au Canada, on assiste à une augmentation continuelle de la population depuis ces dernières dizaines d'années. D'après [2] la population canadienne est estimée à près de 40 millions d'individus en 2023, soit une augmentation de 2.9% par rapport à 2022. Avec l'évolution grandissante de la population, les systèmes de santé sont mis au défi d'accueillir toujours plus de patients. L'enjeu pour nos sociétés est double : garder un accès aux soins de qualité et permettre au plus grand nombre d'accéder aux soins. Ce sont 200 milliards de dollars sur 10 ans qui ont été investis en 2023 pour la santé [3]. En outre, les personnes agées sont celles qui sont les plus susceptibles de requérir des soins. Cette tranche de la population préfère aussi vieillir à domicile plutôt qu'en hôpital. Or, [4] estime qu'en 2037 la part des Canadiens de plus de 65 ans aura augmenté de 37% par rapport à 2023. Le don de soins à domicile apparaît donc comme un système à fort potentiel pour résoudre les problèmes de logistiques et d'engorgement hospitalier liés à l'augmentation de la population.

La recherche en optimisation en santé s'est intensifiée et ces dernières années de nouvelles approches de résolution de problèmes d'optimisation (dénommés *COP* pour *Combinatorial Optimization Problems*) ont été développées. Parmi elles figure l'approche hybride d'intégration de *Reinforcement Learning* (*RL*) à des algorithmes d'optimisation combinatoire. Cette approche est désignée dans [5] par *Neural Combinatorial Optimization*. L'intégration de techniques de *RL* à des méthodes de recherche locale cherche à bénéficier de la robustesse de la recherche locale et du caractère adaptatif des modèles de *RL*. Dans ce contexte, ce mémoire porte sur l'intégration de *RL* à la métaheuristique *Adaptive Large Neighborhood Search* (*ALNS*) dans le cadre de la résolution d'un problème de planification et d'ordonnancement de soins à domicile.

A travers ce mémoire, nous nous questionnons sur la capacité d'un modèle de *RL* intégré à *ALNS* d'apporter de meilleurs résultats en terme de qualité et de temps. Cette question met en lumière l'enjeu lié à la capacité d'un modèle de *RL* à appréhender les dynamiques du problème de panification et d'ordonnancement étudié. Pour répondre à cette question, nous avons suivi une approche en trois parties. Dans une première partie, nous effectuons une revue de littérature sur les travaux scientifiques qui ont combiné un modèle de *RL* à un algorithme de recherche locale et plus particulièrement à la métaheuristique *ALNS*. Dans une deuxième partie, nous nous appuyons sur cette revue de littérature pour intégrer du *RL* à *ALNS*. La méthode employée est de remplacer la partie adaptative de *ALNS* sur la sélection des opérateurs par un modèle de *RL*. Dans l'optique de développer le modèle qui

est le plus performant, nous configurons deux modèles qui ont deux modes d'apprentissage différents. Le premier modèle est *ALNS-PPO*, il utilise l'algorithme *Proximal Policy Optimization* (*PPO*) qui est *on-policy*. Le second modèle est *ALNS-SAC*, il utilise l'algorithme *Soft Actor-Critic* (*SAC*) qui est *off-policy*. Dans une troisième partie, nous évaluons les performances de *ALNS-PPO* et *ALNS-SAC* relativement à *ALNS* sur un ensemble de données que nous avons générées.

Nous avons configuré *ALNS-PPO* et *ALNS-SAC* de telle sorte qu'ils vivent dans le même environnement, c'est-à-dire qu'ils sont configurés avec les mêmes vecteurs d'observation, les mêmes vecteurs d'actions et la même fonction de récompense. Ils sont aussi configurés dans leur version discrète. Les caractéristiques d'observations utilisées dans cette étude sont composées de deux catégories. La première catégorie est constitutée d'observations dépendantes du problème et comporte le nombre de patients et le nombre d'aides-soignants du problème. La deuxième catégorie est celle des observations indépendantes du problème et parmi elles figurent les observations de performance de recherche, comme par exemple les coûts de la meilleure solution, de la solution courante et de la solution candidate, et les observations liées à l'historique de la recherche, comme la dernière action sélectionnée par le modèle et l'itération. Une action est un vecteur multidimensionnel composé d'un opérateur de destruction, d'un opérateur de réparation et d'un degré de destruction de la solution par l'opérateur de destruction. La fonction de récompense utilisée est une fonction à valeur réelle et se définissant comme la valeur relative du coût de la solution courante par rapport à la meilleure solution.

Nous avons en outre généré un ensemble de données en se basant sur des données réelles fournies par Alayacare. La génération a été faite en faisant varier comme paramètres d'instance le nombre de patients, le nombre d'aides-soignants, le nombre d'expertises maximales des patients, le type de génération des localisations et les pourcentages de visites préprogrammées laissées fixes. La phase de testes a consisté en la comparaison des performances en qualité de solution et en temps des algorithmes *ALNS-PPO* et *ALNS-SAC* à *ALNS*. Les résultats des comparaisons indiquent que *ALNS* surpasse *ALNS-PPO* et *ALNS-SAC* sur l'ensemble des données étudiées. Les solutions de *ALNS* sont meilleures en qualité et obtenues en un temps inférieur. En outre, le résultats montrent que *ALNS* a su mieux généraliser aux variations des paramètres d'instance.

Cette étude a été menée en collaboration avec l'entreprise Alayacare. Alayacare est une entreprise fondée en 2014 et développe une plateforme de gestion de tâches de travail pour les compagnies d'aide à domicile.

Ce mémoire est constitué des chapitres suivants. Le chapitre 2 propose une revue de littérature

des études qui ont proposé d'intégrer du *RL* à un algorithme d'optimisation combinatoire, et en particulier à l'algorithme *ALNS*. Le chapitre 3 explicitera l'approche utilisée afin de répondre aux objectifs fixés du mémoire. Dans ce chapitre sera présentée la conception de l'architecture de *RL* combinée à l'*ALNS*. Dans le chapitre 4 nous évaluerons les performances des modèles par rapport à *ALNS* à partir d'un ensemble de données qu'on aura générées. Enfin une conclusion sera apportée dans le chapitre où nous ferons une synthèse des travaux réalisés, porterons un regard critique sur la solution proposée et suggérerons des pistes d'amélioration.

CHAPITRE 2 REVUE DE LITTÉRATURE

L'objectif de cette revue de littérature est de se mettre à jour sur le paysage des méthodes d'hybridation utilisées entre des techniques de *RL* et des algorithmes de recherche locale. Nous étudions d'abord des travaux plus généraux mettant en relation du *RL* à des algorithmes d'optimisation traditionnels. Le but de cette première partie est de mieux appréhender les différentes formes d'intégration de *RL* aux *COP*. Dans une seconde partie, nous proposons d'étudier différentes méthodes d'hybridation entre des techniques de *RL* et des algorithmes de recherche locale, en mettant l'accent sur la métaheuristique *ALNS*. Le but de cette deuxième partie est d'avoir une connaissance approfondie des techniques d'intégration de *RL* à *ALNS* et de s'en servir comme base pour notre projet.

2.1 Intégration de *RL* dans la résolution de *COP*

Dans cette partie nous proposons une revue de littérature qui expose des travaux qui ont développé des approches hybrides de résolution de *COP* en mettant en relation du *RL* à des algorithmes d'optimisation traditionnels. Dans un premier temps, nous étudions la motivation d'intégrer du *RL* aux *COP*, puis nous proposons une revue des travaux donc certains ont combiné du *RL* à des algorithmes d'optimisation, tandis que d'autres ont remplacé complètement des algorithmes d'optimisation par des méthodes de *RL*.

2.1.1 Motivation d'intégrer du *RL* aux *COP*

Généralement, les *COP* étudiés dans la littérature sont de complexité *NP*-difficile (*NP* pour *Non-deterministic Polynomial*) dont la résolution exige de grandes capacités de calcul. Cette caractéristique a incité les chercheurs à se tourner vers de nouvelles approches de recherche plus efficaces. L'apport du *RL* dans le développement de nouvelles techniques de résolution a permis d'augmenter les performances des algorithmes d'optimisation traditionnels [6–8]. Dans [5] est proposé le *Neural Combinatorial Optimization* (*NCO*) pour mentionner l'ensemble des approches de *RL* utilisées dans le contexte de la résolution de *COP*. Parmi les approches de recherche qui combinent des méthodes de *RL* à des algorithmes de résolution de *COP* on en retouve deux principales : l'intégration d'une couche de *RL* dans les algorithmes de *CO* et le développement de méthodes de *RL* qui remplacent les algorithmes d'optimisation traditionnels.

2.1.2 *RL* combiné à un algorithme d'optimisation

Ce paragraphe présente des travaux qui cherchent à résoudre un *COP* avec un algorithme d'optimisation traditionnel auquel a été ajouté un module de *RL*. Les auteurs de [9–11] intègrent du *RL* à des algorithmes de Branch & Bound (*B&B*) afin d'améliorer les décisions de branchement. [9] utilise l'apprentissage par imitation pour apprendre à un modèle à imiter le comportement d'un expert lors du choix de l'expension des noeuds durant l'algorithme. Deux politiques sont entraînées : une politique de sélection qui choisit un noeud à étendre et une polique d'élagage de noeud qui décide de si le noeud choisi vaut la peine d'être élagué. Les auteurs de [10] entraînent quand à eux un modèle de sélection de variables de branchement appelé *TreeGate*. Leur étude porte sur la paramétrisation de l'espace sous-jacent des arbres de recherche de *B&B*. Dans [11] est construit un modèle de sélection adaptative de plans sécants. Les auteurs développent un réseau *LSTM* qui encode les contraintes reliées au problème étudié et l'espace des actions correspond à l'ensemble des plans sécants possibles à l'itération courante.

2.1.3 *RL* pour remplacer des algorithmes d'optimisation

Dans d'autres travaux l'approche suivie est de remplacer intégralement un algorithme d'optimisation par un modèle de *RL* qui résout le même problème. Les auteurs de [12–14] proposent une approche de résolution directe de graphes en combinant du *graph embedding* à du *RL*. Dans [12] est présentée une heuristique d'apprentissage pour résoudre directement les problèmes de couverture minimale, de coupe maximale et le *Travelling Salesman Problem* (*TSP*). Le *graph embedding* est employé afin de transformer la structure initiale en graphe en structure vectorielle. La structure vectorielle est ensuite utilisée pour l'apprentissage d'un *Q-network*. Les auteurs de [13] proposent un modèle de *GCN* qui imite un algorithme de *B&B* afin de résoudre directement des *COP*. Le modèle apprend une politique de sélection de variables par imitation des décisions d'un expert. Dans [14], les auteurs propsent un modèle *GCN* récurrent pour la résolution d'un problème de *SAT* encodé en graphe. Les auteurs de [5, 15, 16] proposent des modèles de *pointer network*. Cette technique d'apprentissage supervisé est introduite par [17] et fait intervenir un auto-encodeur qui génère une séquence de pointeurs (indices) correspondant aux positions de la séquence d'entrée. Les auteurs de [15] utilisent cette aproche pour la résolution du problème du sac-à-dos 0-1. Leur auto-encodeur est composé de deux *RNN* et des échantillons de problèmes du sac-à-dos sont générés aléatoirement puis labellisés via leur résolution par programmation dynamique. Les auteurs de [5] mettent en jeu deux *RNN* comme auto-encodeur pour résoudre le *TSP*. L'algorithme d'apprentissage utilisé se base sur l'algorithme *Asynchronous Advantage Actor Critic* [18] (*A3C*). En outre,

les auteurs de [16] proposent une combinaison d'un *pointer network* avec un *Q-network* pour résoudre le *TSP*.

2.2 Intégration de *RL* à *ALNS*

Ainsi, il existe des manières variées d'intégrer des méthodes de *RL* à des *COP*. Parmi les travaux considérés cans cette revue, certains se sont intéressés à l'intégration de *RL* à des algorithmes de recherche locale. Le prochain paragraphe traite de travaux qui ont combiné des techniques de *RL* à la métaheuristique *LNS*. D'autres études se sont intéressées plus particulièrement à la métaheuristique *ALNS*. Dans un deuxième paragraphe, nous présenterons *ALNS*, puis nous étudierons dans un troisième paragraphe les études qui ont intégré du *RL* à l'algorithme *ALNS*.

2.2.1 *RL* et méthodes de recherche locale

Les auteurs de [19] présentent une extension de la méthode *LNS* [20, 21], la *Neural Large Neighborhood Search*. Leur approche consiste à développer un modèle qui apprend à détruire une solution. Ils utilisent un *Graph Neural Networks* (*GNN*) pour paramétriser la politique et le critique. Par conséquent, le modèle apprend à choisir quelles sont les variables de la solution qui doivent être éliminées. Après la phase de destruction, la solution est réparée à l'aide d'un solveur de programmation en nombres entiers mixtes. De même, les auteurs de [22] développent une combinaison entre la méthode *LNS* et des techniques de *RL*, mais ils se focalisent sur l'opérateur de réparation. Ils construisent un réseau de neurones en acteur-critique qui est entraîné à choisir l'opérateur de construction. Leur méthode, appliquée au *Vehicle Routing Problem with Time Windows* (*VRPTW*), a une bonne capacité à généraliser aux instances de plus grande taille.

2.2.2 Adaptive Large Neighborhood Search

L'algorithme *ALNS* est une métaheuristique reposant sur le principe de recherche locale. Cette famille de métaheuristiques se base sur la modification itérative d'une solution en explorant son voisinage. La méthode *ALNS* est une extension de la méthode *LNS* dont la stratégie d'exploration de l'espace des solutions s'adapte dynamiquement à la progression de la recherche. L'adaptation a lieu au niveau de la sélection des opérateurs. *ALNS* opère des grandes modifications à la solution, via la nature et le diversité des opérateurs employés, ce qui permet d'explorer un plus large voisinage. La diversité des voisinages explorés permet à *ALNS* de sortir des minima locaux. L'efficacité de la méthode *ALNS* dans sa recherche

de solutions fait d'elle une méthode utilisée dans de nombreuses études récentes [23–29]. Cette méthode a apporté des résultats significatifs dans la résolution des *COP*, notamment le *VRP* [30].

2.2.3 Intégration de *RL* à *ALNS*

La revue de littérature nous a permis de mettre en relation quatre études [31–34] qui intègrent du *RL* à *ALNS*. Nous les avons comparé suivant les choix que les auteurs ont fait par rapport aux quatre axes suivants : l'architecture du modèle, l'espace des observations, l'espace des actions et le système de récompense.

Architecture du modèle Les auteurs de [31–33] développent un *Multilayer perceptron* (*MLP*) entraîné avec l'algorithme *on-policy PPO*. Par contraste, les auteurs de [34] exploitent un *GNN* en utilisant l'algorithme *off-policy* de *Q-learning*. Ces derniers motivent leur choix d'une structure en graphe par le fait qu'elle peut par nature capturer des *patterns* indépendants de la taille du problème et donc de généraliser plus facilement aux grandes instances.

Espace des observations Chacune des études décompose l'espace d'observation en deux ensembles. Le premier contient les caractéristiques dépendantes du problème et le deuxième contient celles qui ne le sont pas. Pour les caractéristiques indépendantes, on retouve généralement les mêmes à travers les travaux. Il s'agit par exemple du coût des solutions courante, candidate et de la meilleure solution, si la solution candidate est meilleure que la solution courante et / ou meilleure que la meilleure solution, ainsi que le nombre d'itérations sans amélioration de la meilleure solution. En outre, [31, 33] n'utilisent que les caractéristiques indépendantes du problème et motivent ce choix par une abilité du modèle à généraliser à d'autres problèmes. En ce qui concerne les variables dépendantes, [32] inclut notamment des variables sur l'équité au sein du personnel hospitalier. Dans [34], les auteurs incluent dans l'observation une variable représentant le budget alloué au choix des opérateurs. L'épisode prend fin une fois que le budget a été entièrement consommé.

Espace des actions Chacune des études considérées inclue naturellement dans l'espace des actions le choix sur les opérateurs de destruction et construction. La distinction se fait sur la manière d'intégrer le degré de sévérité sur l'opérateur de destruction à l'espace des actions. [31, 32] intègrent ce paramètre en ajoutant à plusieurs reprises le même opérateur à l'espace des actions mais avec chaque fois un degré de sévérité différent. Par contraste, [33] définit le degré de sévérité comme une autre composante de l'action et ajoute aussi à celle-ci la

valeur de la température utilisée dans le recuit simulé. En outre, les auteurs de [34] définissent le degré de sévérité comme un hyperparamètre du modèle et ils font une recherche en grille pour déterminer quelle est sa valeur optimale. Par ailleurs, la taille de l'espace d'action peut varier considérablement d'une étude à une autre. [31] emploie 142 couples d'opérateurs différents au total tandis que [34] se limite à 28 couples.

Système de récompense Dans [31–33], la fonction de récompense est définit sur le même principe que la fonction de score de *ALNS* et peut s'écrire sous une forme générale donnée par l'équation (2.1).

$$R = \begin{cases} r_0 & \text{si la solution candidate est moins bonne que} \\ & \text{la solution courante et elle n'est pas acceptée} \\ r_1 & \text{si la solution candidate est moins bonne que} \\ & \text{la solution courante et elle est acceptée} \\ r_2 & \text{si la solution candidate est meilleure que} \\ & \text{la solution courante} \\ r_3 & \text{si la solution candidate est meilleure que} \\ & \text{la meilleure courante} \end{cases} \tag{2.1}$$

$$\text{avec } r_0 \leq r_1 \leq r_2 \leq r_3$$

Dans [33] la récompense est définit par $r_0 = r_1 = r_2 = 0$ et $r_3 = 5$, tandis que [32] la définit par $r_0 = 0, r_1 = 1, r_2 = 3, r_3 = 5$. Par contraste, [34] récompense plutôt le modèle proportionnellement à l'amélioration du coût par rapport à la solution initiale. En outre, [31] utilise la même fonction de récompense que [31] et la compare à une autre fonction qui calcule le coût relatif de la solution candidate à celui de la meilleure solution. Dans le cas de cette étude c'est la première fonction qui apporte de meilleurs résultats.

Cette revue de littérature montre qu'intégrer du *RL* à la sélection des opérateurs de *ALNS* était le choix de référence parmi les travaux étudiés. Par la même occasion, cette revue montre que les choix de conception d'une telle méthode sont très variés. Nous proposons dans le prochain chapitre d'intégrer du *RL* à *ALNS* à partir des informations acquises dans cette revue de littérature.

CHAPITRE 3 MÉTHODOLOGIE

Ce chapitre est consacré à la présentation de la méthode choisie pour intégrer du *Reinforcement Learning* (*RL*) à *ALNS*. Une première partie est dédiée à la présentation et à l'explication des termes importants issus du *RL*. Dans une deuxième partie est présenté la fonctionnement de la méthode *ALNS*. Nous présenterons notamment l'ensemble des opérateurs utilisés ainsi que la fonction de coût mise en jeu. Dans une troisième partie nous explicitons la méthode de *RL* utilisée. Enfin, nous présentons les deux algorithmes pour l'apprentissage du modèles.

3.1 Reinforcement Learning

L'apprentissage par renforcement, ou *Reinforcement Learning* (*RL*), est une branche du *Machine Learning* (*ML*) qui consiste en l'association, par un modèle mathématique, d'un ensemble d'états à un ensemble d'actions dans le but de maximiser des récompenses cumulées. Cette partie fournit une présentation générale des concepts du *RL* avant de donner quelques présentées certaines notions théoriques de cette branche de mathématiques.

3.1.1 Concept

En *RL*, le but est de concevoir un agent (le modèle) qui choisit des actions qui maximisent les récompenses cumulées reçues. La figure 3.1 illustre l'intéraction entre un agent et son environnement.

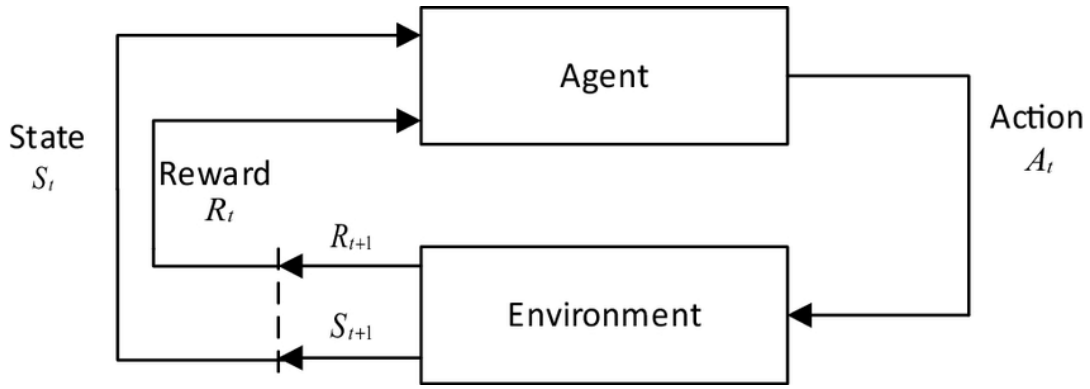


FIGURE 3.1 Représentation schématique d'un système de *RL*. Illustration tirée de [1]

L'intération est un processus qui se répète de manière itérative. A chaque itération, l'agent

observe son environnement, choisit une action à partir de cette observation, transite dans l'état suivant et reçoit en retour une récompense. Des exemples populaires d'applications du *RL* sont la robotique [35] ou l'industrie vidéoludique [36].

3.1.2 Formalisme

Un système de *RL* peut se décrire comme un processus de décision de Markov (*MDP*) évoluant dans un environnement $\mathcal{E}$. Le *MDP* (i.e. l'agent) est défini par le tuple $(S, A, R, T, \gamma, \pi)$ où :

. S est l'espace des états
. A est l'espace des actions
. $R : S \times A \times S \to \mathbb{R}^d$ est la fonction de récompense, $d \geq 1$
. $P : S \times A \times S \to \mathbb{R}$ est la distribution de probabilité de transition
. $\gamma \in [0, 1)$ est le facteur de dévaluation (*discount factor*)
. $\pi : S \to A$ est la politique de l'agent

L'interaction entre l'agent et son environnement $\mathcal{E}$ est indexée par un pas de temps $t \in \mathbb{N}$. A chaque pas t, l'agent observe un état $s_t \in S$ et choisit une action $a_t \in A$ à partir de s_t et selon sa politique π. L'action a_t appliquée à l'état s_t fait transiter l'agent de l'état s_t vers l'état $s_{t+1} \sim P(\ . \mid s_t, a_t)$ selon la distribution de probabilité de transition P. La récompense $\mathbf{r}_t = R(s_t, a_t, s_{t+1}) \in \mathbb{R}^d$ est calculée et correspond à la performance de l'agent vis-à-vis de l'atteinte des objectifs. Dans le cas où $d > 1$, le sytème devient un *Multi-Objective Markov Decision Process* (*MOMDP*) [37].

La politique π détermine le comportement de l'agent en associant l'ensemble des actions A à l'ensemble des états S. Dans ce qui suit, nous nous plaçons dans le cas où la politique π est paramétrée par θ et la noterons π_θ. Lorsque π_θ est déterministe, l'action a_t est donnée par la formule déterministe $a_t = \pi_\theta(s_t)$. Nous nous placerons dans la suite dans le cas où π_θ est stochastique, c'est-à-dire quand le choix de l'action a_t connaissant s_t est une probabilité paramétrée par θ et on note alors $a_t \sim \pi_\theta(\ . \mid s_t)$.

On note l'état inital par s_0 et sa distribution par ρ_0. Une trajectoire τ associée s'écrit $\tau = (s_0, a_0, s_1, a_1, ...)$. En notant $T \in \mathbb{N} \cup \{+\infty\}$ la longueur de la trajectoire τ, la probabilité d'obtenir τ est donnée par l'équation (3.1).

$$p(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t) \tag{3.1}$$

Le retour devalué (*discounted return*) associé à π est la somme des récompenses obtenues par

l'agent sur toute la trajectoire et dévaluées par le paramètre γ. Sa formule est donnée par l'équation (3.2).

$$G(\tau) = \sum_{t=0}^{T} \gamma^t \cdot \mathbf{r}_t \tag{3.2}$$

En *RL*, le but est de trouver la politique optimale qui maximise l'espérance du retour dévalué. On note cette valeur $J(\pi)$ et sa formule est donnée par l'équation (3.3).

$$J(\pi) = \int_{\tau} p(\tau \mid \pi) G(\tau)\, d\tau = \mathbb{E}_{\tau \sim \pi}[G(\tau)] \tag{3.3}$$

La formule de la politique optimale π^* est donnée par l'équation (3.4).

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{3.4}$$

Lorsque l'agent apprend une politique optimale qui est celle utilisée pour collecter les expériences, on dit que l'apprentissage est *on-policy*. A l'inverse, lorsque l'agent apprend une politique optimale indépendemment de ses choix, on dit que l'apprentissage est *off-policy*.

3.2 Adaptive Large Neighborhood Search

Maintenant que les éléments clés associés du *RL* ont été posés, nous nous intéressons à l'architecture de l'algorithme *ALNS*. Cette partie se consacre donc à donner une explication du fonctionnement de *ALNS* afin de mieux pouvoir cerner les possibilités d'intégration de techniques de *RL* à cette métaheuristique. En particulier, on s'attachera à décrire la partie adaptative de la méthode *ALNS*. Dans cette étude, nous utiliserons l'algorithme *ALNS* développé dans [38].

3.2.1 Fonctionnement de *ALNS*

La métaheuristique *ALNS* appartient à la famille des métaheuristiques de recherche locale. Le principe de cette famille d'algorithmes est d'améliorer itérativement une solution, appelée solution courante, en recherchant dans son voisinage. La nouvelle solution ainsi trouvée est appelée la solution candidate. La meilleure solution désigne la solution de coût minimal parmi toutes les solutions visitées depuis le début de la recherche.

Dans toutes la suite, nous noterons la solution courante, la solution candidate et la meilleure solution par $x\,, x'$ et x^* respectivement. On notera par x_t la solution au pas t de l'algorithme. On notera aussi par f la fonction coût.

Le pseudocode générique d'un algorithme de recherche locale est donné dans la figure 3.2. Nous décrivons ci-après le fonctionnement de ce pseudocode. La solution courante et la meilleure solution sont initialisées comme étant la solution initiale. Tant que le critère d'arrêt n'est pas atteint, on effectue un pas de recherche. Un pas de recherche correspond à la sélection d'un voisin de la solution courante nommé la solution candidate. Si la solution candidate est acceptée par le critère d'acceptation, alors la la solution courante devient la solution candidate. Ensuite, la meilleure solution devient la solution courante si cette dernière est de meilleure qualité. L'algorithme renvoie en sortie la meilleure solution.

Algorithm 1 Recherche Locale

Require: Solution initiale x_0, Critère d'acceptation C
Ensure: Meilleure solution trouvée x^*

```
1: function RechercheLocale(x_0, C)
2:     Initialiser x ← x_0
3:     Initialiser x* ← x_0
4:     while le critère d'arrêt n'est pas vérifié do
5:         x' ← sélectionner un voisin de x
6:         if C(x, x') then
7:             x ← x'
8:             if f(x) < f(x*) then
9:                 x* ← x
10:            end if
11:        end if
12:    end while
13:    return x*
14: end function
```

FIGURE 3.2 Pseudocode de la recherche locale

La métaheuristique de *ALNS* repose sur le principe de recherche locale auquel sont ajoutées les notions d'opérateurs de construction et de destruction, de recherche adaptative et de critère d'acceptation.

A partir de la diverité des opérateurs utilisés, *ALNS* diversifie les voisinages des solutions ce qui lui permet d'échapper aux minima locaux et de continuer à rechercher de meilleures solutions.

A chaque itération, un couple d'opérateur de destruction et de réparation sont choisis pour modifier une partie de la solution courante x_t. Le choix de ces opérateurs est dicté par un mécanisme adaptatif de la recherche. Enfin, le critère d'acceptation est employé pour accepter ou non la solution candidate x'_t comme nouvelle solution courante x_t. Le schéma algorithmique de *ALNS* est donné dans la figure 3.3.

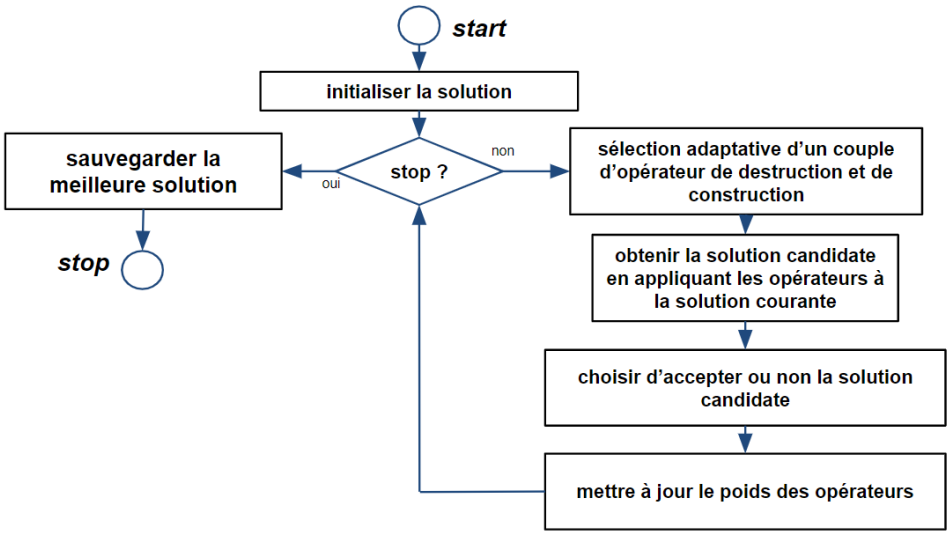


FIGURE 3.3 Schéma du principe de *ALNS*

Le pseudocode de *ALNS* est donné dans la figure 3.4.

Algorithm 2 Adaptive Large Neighborhood Search (ALNS)

Require: Solution initiale x_0, opérateurs de destruction D et de réparation R
Ensure: Meilleure solution trouvée x^*

```
1: function ALNS(x_0, D, R)
2:     x ← x_0, x* ← x_0
3:     W ← InitialiserPoids(D, R)
4:     S ← 0
5:     T ← T_0

6:     while le critère d'arrêt n'est pas vérifié do
7:         Sélectionner un opérateur de destruction d et de réparation r selon W
8:         x' ← Appliquer r ∘ d à x
9:         is_accepted ← AccepterSolution(x, x', T)
10:        γ ← CalculerGamma(x, x', x*, is_accepted)
11:        MettreAJourPoids(W, d, r, S, γ)
12:        S ← S + γ
13:        if is_accepted then
14:            x ← x'
15:        end if
16:        if f(x) < f(x*) then
17:            x* ← x
18:        end if
19:    end while

20:    return x*
21: end function
```

FIGURE 3.4 Pseudo-code de *ALNS*

Le critère d'acceptation est donné par la fonction AccepterSolution dont le pseudo-code est fourni dans l'annexe D. Une description du critère d'acceptation utilisé dans cette étude est faite ultérieurement dans cette section. Le poids sont initialisés par la fonction Initialiser-Poids dont le pseudo-code est fourni dans l'annexe A. La mise à jour des poids est faite par la fonction MettreAJourPoids dont le pseudo-code est fourni dans l'annexe B. Le fonction MettreAJourPoids utilise le retour de la fonction CalculerGamma dont le pseudo-code est fourni dans l'annexe C. Une description détaillée de la mise à jour des poids est donnée est faite ultérieurement dans cette section.

Opérateurs de destruction et de réparation

Ceux-ci sont des heuristiques qui modifient une partie de la solution courante et l'ensemble

des opérateurs P est défini avant l'initialisation de l'algorithme. En outre, l'ensemble des opérateurs P se décompose en deux sous-ensembles : un ensemble d'opérateurs de destruction D et un ensemble d'opérateurs de réparation R. Ainsi, $P = D \cup R$. La solution courante est modifiée selon le principe de *ruin and recreate*. A chaque itération un opérateur de destruction et un opérateur de réparation sont choisis parmi les ensembles D et R respectivement et sont séquentiellement appliqués à la solution courante x_t pour créer la solution candidate x'_t.

Recherche adaptative

La partie adaptative concerne le modification dynamique dans le choix des opérateurs de destruction et de réparation en fonction de leur efficacité au cours de la recherche. Formellement, le mécanisme sous-jacent au choix des opérateurs est un couple de variables aléatoires X_D et X_R qui représentent l'opérateur de destruction (respectivement de réparation) choisi à chaque itération. La distribution de probabilité $\mathcal{D}_{X_D}$ (respectivement $\mathcal{D}_{X_R}$) de X_D (respectivement X_R) est mise à jour périodiquement le long des itérations. L'*ALNS* met aussi en jeu un ensemble de poids des opérateurs W. A chaque opérateur $p \in P$ correspond un poids $w_p \in W$. Le poids w_p de chaque opérateur p rend compte des performances de p dans les itérations passées dans la recherche d'une solution optimale. En outre, un score S est utilisé pour mettre à jour les poids des opérateurs.

Les poids $w \in W$ et le score S sont initialisés selon l'équation (3.5) puis $\mathcal{D}_{X_D}$ et $\mathcal{D}_{X_R}$ sont initialisées selon l'équation (3.6).

$$\begin{aligned} &w_1 = w_2 = ... = w_{|D|} \text{ où } w_i \in D\,, i \in \{\,1, \ldots, |D|\,\} \quad , \\ &w_1 = w_2 = ... = w_{|R|} \text{ où } w_i \in R\,, i \in \{\,1, \ldots, |R|\,\} \\ &S \leftarrow 0 \end{aligned} \tag{3.5}$$

$$\begin{aligned} &X_D \sim \mathcal{D}_{X_D} \text{ avec } \forall p \in D,\ \ p(X_D = p) \leftarrow \frac{w_p}{\sum_{p \in D} w_p} \\ &X_R \sim \mathcal{D}_{X_R} \text{ avec } \forall p \in R,\ \ p(X_R = p) \leftarrow \frac{w_p}{\sum_{p \in R} w_p} \end{aligned} \tag{3.6}$$

X_D (respectivement X_R) est initialement une distribution uniforme sur l'ensemble des opérateurs D (respectivement R). A chaque itération, un couple d'opérateurs $(p \in D,\ p' \in R)$, qu'on note $(p_D,\ p_R)$, est choisi pour modifier la solution. Le score S et les poids $w \in W$ sont ensuite mis à jour itérativement selon l'équation (3.7). Puis les distributions $\mathcal{D}_{X_D}$ et $\mathcal{D}_{X_R}$ sont mises à jour itérativement selon l'équation (3.6).

$$
\begin{aligned}
&\forall p \in D,\ \ w_p \leftarrow \frac{w_p \cdot S + 1_{\{p=p_D\}} \cdot \gamma}{S+\gamma} \\
&\forall p \in R,\ \ w_p \leftarrow \frac{w_p \cdot S + 1_{\{p=p_R\}} \cdot \gamma}{S+\gamma} \\
&S \leftarrow S + \gamma
\end{aligned}
\tag{3.7}
$$

La valeur de γ est une fonction des solutions courante, candidate et de la meilleure solution. Dans cette étude, nous avons choisit comme fonction γ celle utilisée généralement dans la littérature et dont la formule est donnée par l'équation (3.8).

$$
\gamma = \begin{cases}
0 & \text{si solution candidate moins bonne que solution courante et n'est pas acceptée} \\
1 & \text{si solution candidate moins bonne que solution courante et est acceptée} \\
3 & \text{si solution candidate est meilleure que solution courante} \\
5 & \text{si solution candidate est meilleure que meilleure solution}
\end{cases}
\tag{3.8}
$$

Plus un opérateur est lié à l'amélioration de la qualité d'une solution, plus son score augmente. Si un opérateur a un score plus grand que les autres opérateurs, son poids sera augmenté à la prochaine mise à jour. A l'inverse un opérateur qui a score plus petit que les autres opérateurs vera le poids associé à son score diminuer à la prochaine mise à jour. Ainsi, l'*ALNS* a une stratégie d'exploration qui varie dynamiquement au cours de la recherche via l'évolution des probabilités de choix des opérateurs.

Critère d'acceptation

Le critère d'acceptation détermine si la solution candidate doit être acceptée comme la solution courante. Ce critère peut reposer sur différents types de mécanismes. Par exemple, une solution peut être acceptée uniquement si elle améliore la solution actuelle. Le critère d'acceptation cherche à équilibrer l'exploration et l'exploitation de l'espace des solutions. Il offre une alternative pour sortir des optima locaux et de diversifier la recherche de nouvelles régions de l'espace.

Dans cette étude, nous utilisons l'algorithme *ALNS* de [38] qui emploie le recuit simulé (*SA* pour *Simulated Annealing*) comme critère d'acceptation. Dans la méthode *SA*, on fait diminuer une probabilité p_{SA} le long de la recherche, à l'image de la température d'un recuit. La valeur de p_{SA} donne la probabilité d'accepter une solution candidate qui est de moins

bonne qualité que la solution courante. Ainsi on observe un voisinage plus large en début de recherche et on se restreint à un voisinage plus qualitatif en fin de recherche. Afin de simuler ce recuit, on introduit une température T initialisée à T_0 en début d'algorithme. Celle-ci est utilisée afin de calculer la probabilité p_{SA} selon la formule (3.9).

$$p_{SA} = \exp\Big(- \frac{f(x'_t) - f(x_t)}{T} \Big) \tag{3.9}$$

La méthode *SA* met aussi en jeu un paramètre α qui est appelé communément *cooling schedule* et est généralement défini par $\alpha = 0.99$. Ce paramètre est utilisé pour mettre à jour la température T à chaque itération. La formule associée à la mise à jour est donnée par l'équation (3.10).

$$T \leftarrow \alpha \cdot T, \quad \alpha \in [0, 1] \tag{3.10}$$

L'algorithme du critère d'acceptation est fourni dans l'annexe D. Cet algorithme met à jour la température à partir de l'algorithme fourni dans l'annexe E.

3.2.2 Représentation d'une solution

L'*ALNS* de [38] est utilisé dans un contexte de génération de routes prises par des aides-soigants pour soigner des patients à domicile. Dans ce contexte, une solution x représente un ensemble de routes prises par les aides-soignants sur un horizon temporel donné. Chaque route correspond à un ensemble de visites faites par un aide-soignant à des patients sur une journée de travail. Un service correspond à un ensemble de visites concernant le soin d'un même patient. Celui-ci peut être délivré par plusieurs aides-soignants différents.

3.3 Opérateurs utilisés

Cette partie décrit la nature des opérateurs utilisés mis en jeu dans cette étude. Les modèles tout comme l'algorithme *ALNS* auront accès à cet ensemble d'opérateurs. Une description du fonctionnement des opérateurs est fournie dans les tableaux 3.1 et 3.2.

Tableau 3.1 Description des opérateurs de destruction

Opérateurs de destruction	
Nom	Description
	n : degré de sévérité de destruction (nombre de services à détruire)
Random	Fait une tirage équiprobable d'une route puis un tirage équiprobable de n services de cette route et supprime les services.
Worst	Supprime n services dont la suppression réduit le plus le coût de la solution.
Random Service Visit	Fait un tirage équiprobable de n services de l'ensemble des routes et supprime ces service.
Flexible Service	Idem que *Random Service Visit* mais la probabilité du tirage d'un service augmente avec le nombre de jours disponibles pour ce service.

Tableau 3.2 Description des opérateurs de réparation

Opérateurs de réparation	
Nom	Description
	n : nombre de services à insérer
Tight Time Window	Trie les n services en fonction de la taille de leur fenêtre de temps par ordre décroissant. Insère chaque service par un algorithme *greedy* sur le coût de la solution.
Number of Authorized Routes	Trie les n services en fonction du nombre d'aides-soignants disponibles par ordre croissant. Insère chaque service en l'associant à l'aide-soignant qui minimise le coût de la solution.
Random Service	Fait un tirage équiprobable sur les n services puis opère un algorithme *greedy* sur le service tiré.
Random Service Nurse	Fait un tirage équiprobable sur les n services et sur les aides-soignants puis opère un algorithme *greedy* sur le service tiré pour cet aide-soignant.

3.4 Solution initiale

La solution initiale est construite en appliquant un opérateur de construction à l'ensemble de patients et d'aides-soignants fournis initialement. L'opérateur utilisé est spécifié à l'initialisation de l'algorithme. Dans cette étude, nous utiliserons l'opérateur *Tight Time Window* qui est utilisé par référence par Alayacare lors de la résolution d'instances.

3.5 Coût d'une solution

Une fois que la solution initiale est construite, *ALNS* va chercher à l'améliorer en lui appliquant à chaque itération un opérateur de destruction suivi d'un opérateur de construction. A chaque itération, une nouvelle solution candidate est donc créée. Pour la comparer à la solution courante, il faut avoir accès à son coût. Dans notre étude, le coût d'une solution x_t est noté par le vecteur $\mathbf{f}(x_t)$ de taille $K = 3$. Ses composantes, les sous-coûts, sont notées par $f_k(x)$, $k \in \{1, 2, 3\}$. Pour comparer deux solutions, on établit une hiérarchie entre les sous-coûts. A chaque composante de $\mathbf{f}(x)$ est associé un degré de hiérarchie. Les degrés de hiérarchie sont définis à l'initialisation de l'algorithme. La relation de supériorité entre deux solutions est définie comme la relation de dominance de Pareto donnée par l'équation (3.11).

$$\begin{aligned} x \preceq x' &\iff \exists\, k_0 \in \{1, \ldots, K\},\ \forall k \in \{1, \ldots, k_0 - 1\},\ f_k(x) = f_k(x') \text{ et } f_{k_0}(x) \geq f_{k_0}(x') \\ x \prec x' &\iff \exists\, k_0 \in \{1, \ldots, K\},\ \forall k \in \{1, \ldots, k_0 - 1\},\ f_k(x) = f_k(x') \text{ et } f_{k_0}(x) > f_{k_0}(x') \end{aligned} \tag{3.11}$$

Le tableau 3.3 fournit une description des composantes du coût d'une solution.

Tableau 3.3 Description des composantes du coût d'une solution

Pénalités du coût	
Nom	Description
Visites rejetées	Nombre de services qui ne sont pas assignés.
Fidélité	Augmente quand la continuité du soin (même aide-soignant par patient) diminue.
Distance	Distance totale parcourue par les aides-soignants.

Nous avons configuré la hiérarchie entre les composantes du coût selon deux scénarios différents, nommés $S1$ et $S2$. Dans le $S1$, l'ordre hiérarchique descendant est : Visites rejetées, Fidélité, Distance. Dans le $S2$, l'ordre hiérarchique descendant est : Visites rejetées, Distance, Fidélité. On a choisi de garder les Visites rejetées en premier car le but recherché en premier est de diminuer au maximum les visites qui ne sont pas planifiées.

3.6 Intégration de Reinforcement Learning à *ALNS*

En ayant une connaissance du fonctionnement de *ALNS* et ayant les outils de *RL* à disposition, nous nous appliquons dans cette partie à développer un modèle de *RL* que nous

intégrerons à *ALNS*. Notre approche est, à l'instar des travaux présentés dans le revue de littérature et qui ont intégré du *RL* à *ALNS*, de remplacer la partie adaptative de *ALNS* par un modèle de *RL*. Ces travaux ont montré que les approches par des modèles *on-policy* comme *off-policy* étaient possibles. Chacun de ces types d'aprentissage a ses forces et faiblesses. Par exemple, l'apprentissage *on-policy* est souvent plus stable parce que la politique entraînée est aussi celle qui explore l'environnement. Cependant, les modèles demandent alors plus d'exploration pour être optimaux. D'un autre côté, l'apprentissage *off-policy* est plus efficace en ce qui concerne l'échantillonnage car la politique peut s'entraîner sur d'anciennes expériences. Par contre, la politique est aussi moins stable car ce n'est pas elle qui collecte les expériences. Ces deux types d'apprentissage sont complémentaires et la réussite de l'un par rapport à l'autre dépend du problème posé. Par conséquent, nous avons choisi de développer deux modèles : *ALNS-PPO* qui est *on-policy* et *ALNS-SAC* qui est *off-policy*. Ainsi, nous augmentons nos chances d'obtenir une méthode plus performante que *ALNS*. *ALNS-PPO* et *ALNS-SAC* auront la même architecture et navigueront dans le même environnement, c'est-à-dire que l'espace d'observation, l'espace d'action ainsi que la fonction de récompense seront les mêmes pour chacun d'eux. Leur différence résidera dans l'algorithme d'apprentissage utilisé. Nous utiliserons l'algorithme *Proximal Policy Optimization* (*PPO*) [39] pour *ALNS-PPO* et l'algorithme *Soft Actor-Critic* (*SAC*) [40] pour *ALNS-SAC*.

Dans cette partie nous expliciterons séquentiellement le principe de la méthode utilisée, l'architecture des modèles, l'ensemble des observations, l'ensemble des actions et le système de récompense mis en jeu pour les modèles. Ensuite, nous présenterons les fonctionnement des modèles *ALNS-PPO* et *ALNS-SAC*.

3.6.1 Principe de la méthode

La figure 3.5 donne le schéma algorithmique de l'approche suivie pour intégrer un modèle de *RL* à *ALNS*. La partie adaptative de *ALNS* est remplacée par un modèle de *RL* qui sélectionne selon son propre mécanisme les opérateurs de destruction et de construction. Le modèle est un réseau de neurones qui prend en entrée une représentation de l'état de recherche de l'itération actuelle et renvoie en sortie une action composée d'un couple d'opérateur de destruction et de réparation et d'un pourcentage de destruction. L'action est ensuite utilisée pour modifier la solution courante.

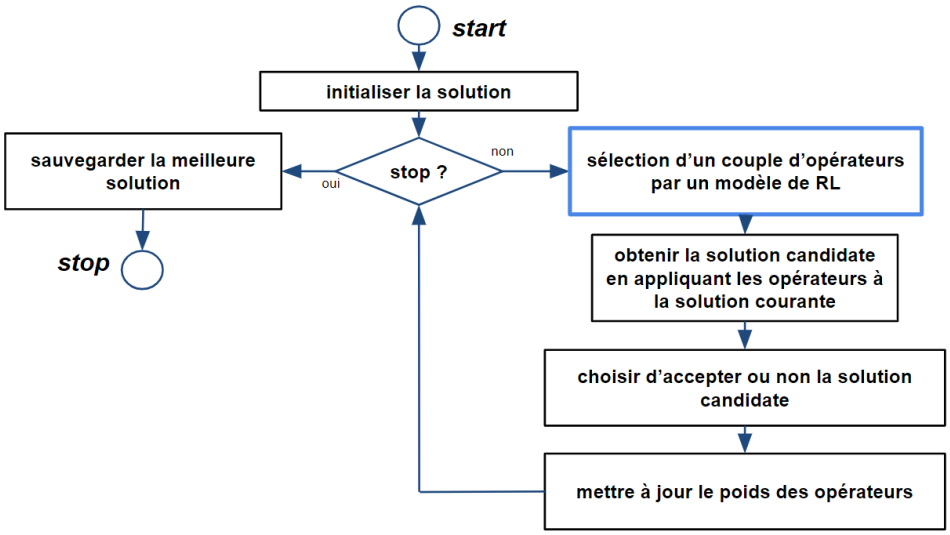


FIGURE 3.5 Schéma algorithmique de l'intégration d'un modèle de *RL* à *ALNS*

Le pseudo-code de l'intégration de *RL* à *ALNS* est donné dans la figure 3.6.

Algorithm 3 *ALNS-RL*

Require: Solution initiale x_0, opérateurs de destruction D et de réparation R, modèle de RL A
Ensure: Meilleure solution trouvée x^*

function *ALNS-RL*(x_0, D, R, A)
    $x \leftarrow x_0$, $x^* \leftarrow x_0$
    $W \leftarrow$ InitialiserPoids(D, R)
    $S \leftarrow 0$
    $T \leftarrow T_0$
    $\theta \leftarrow$ InitialiserPoidsduModèle(A)

    while le critère d'arrêt n'est pas vérifié **do**
        Sélectionner un opérateur de destruction d et de réparation r selon A
        $x' \leftarrow$ Appliquer $r \circ d$ à x
        $is\_accepted \leftarrow$ AccepterSolution(x, x', T)
        if il est temps de mettre à jour les poids du modèle A **then**
            MettreAJourRLPoidsModele(θ, A)
        end if
        if $is\_accepted$ **then**
            $x \leftarrow x'$
        end if
        if $f(x) < f(x^*)$ **then**
            $x^* \leftarrow x$
        end if
    end while

    return x^*
end function

FIGURE 3.6 Pseudo-code de l'intégration de *RL* à *ALNS*

Les fonctions InitialiserPoidsduModèle et MettreAJourRLPoidsModele sont spécifiques à l'algorithme de *RL* associé à *ALNS*. Dans notre cas, les algorithmes de *RL* considérés sont *PPO* et *SAC* et les modèles associés sont *ALNS-PPO* et *ALNS-SAC*. Le fonctionnement de chacun des modèles sera spécifié ultérieurement.

3.6.2 Architecture du modèle

L'architecture du modèle est celle d'un *MLP* dont les poids sont notés θ. La politique π du modèle est notée π_θ. Le modèle prend en entrée une représentation vectorielle de l'observa-

tion. Il renvoie en sortie une représentation vectorielle de l'action. La figure 3.7 représente l'architecture du modèle.

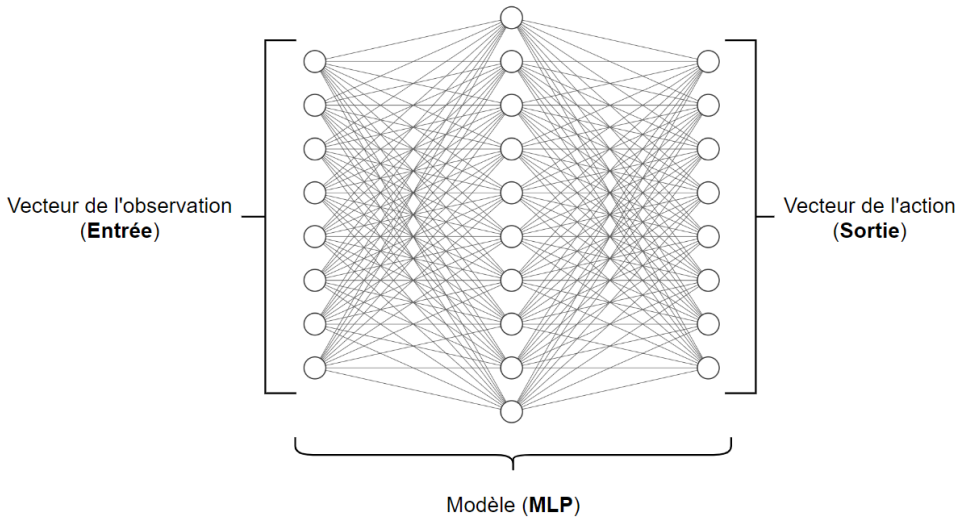


FIGURE 3.7 Architecture en *MLP* du modèle, exemple avec une couche cachée

3.6.3 Observations

Une observation (ou état) $s_t \in S$ est un vecteur contenant des caractéristiques d'observation (*features*) de la solution et du problème au pas de temps t. Pour le choix des variables d'observation, nous nous sommes principalement basés sur les travaux étudiés dans la revue de littérature. Le tableau 3.4 regroupe les variables que nous avons intégrées au vecteur d'observation.

TABLEAU 3.4 Caractéristiques d'observation trouvées pertinentes pour caractériser l'observation

Caractéristiques d'observations	
Notation	Description
n_P	nombre de patients de l'instance
n_A	nombre d'aides-soignants de l'instance
$f(x_t)$	coût de la solution courante x_t
$f(x'_t)$	coût de la solution candidate x'_t
$f(x^*_t)$	coût de la meilleure solution x^*_t
Δf_t	écart en coût entre x_t et x_{t-1}
Δf^*_t	écart en coût entre x_t et x^*_t
$1_{\{x'_t \succ x_t\}}$	vaut 1 si $x'_t \succ x_t$, 0 sinon
$1_{\{x'_t \succ x^*_t\}}$	vaut 1 si $x'_t \succ x^*_t$, 0 sinon
$1_{\{x'_t \prec x_t\,,\,\text{acc}\}}$	vaut 1 si $x'_t \prec x_t$ et x'_t est acceptée, 0 sinon
$1_{\{x'_t \prec x_t\,,\,\overline{\text{acc}}\}}$	vaut 1 si $x'_t \prec x_t$ et x'_t n'est pas acceptée, 0 sinon
t	pas de la résolution
$\boldsymbol{a}_{t-1}$	dernière action en *one hot* encodage
T	température du recuit simulé
n_{seen}	nombre de solutions différentes vues depuis l'initialisation
$n_{\text{last\_best}}$	nombre de pas depuis la dernière amélioration de x^*_t

La variable Δf_t permet au modèle de savoir s'il se dirige dans une bonne direction, tandis que la variable Δf^*_t lui permet de savoir à quel point x_t est meilleure que x^*_t. Les variables $f(x_t), f(x'_t)$ et $f(x^*_t)$ sont utiles au modèle pour déterminer où il se situe dans l'avancement de la recherche en terme de qualité de solution. Leur caractère de coût en valeur *absolue* vient complémenter les variables Δf_t et Δf^*_t qui représentent un écart relatif. Les variables $1_{\{x'_t \succ x_t\}}, 1_{\{x'_t \succ x^*_t\}}, 1_{\{x'_t \prec x_t\,,\,\text{acc}\}}$ et $1_{\{x'_t \prec x_t\,,\,\overline{\text{acc}}\}}$ permettent au modèle de connaître la qualité relative de la solution x'_t par rapport à x_t et x^*_t. Le pas t permet au modèle de se situer par rapport à l'avancement de la recherche. Le but de la variable $\boldsymbol{a}_{t-1}$ est d'être mise en relation avec Δf_t pour signaler au modèle si la dernière action a mené à une amélioration de la solution ou non. Le but derrière l'ajout de la température T du recuit simulé est de permettre au modèle d'adapter sa stratégie en fonction de la probabilité de la solution x_t d'être remplacée par une moins bonne solution. En outre, la variable n_{seen} a pour but de signaler au modèle quand il est préférable de favoriser l'exploration ou l'intensification. Enfin la variable $n_{\text{last\_best}}$ apporte une information complémentaire par rapport à la variable n_{seen}.

3.6.4 Actions

Nous avons choisi de construire un vecteur d'action multi-dimensionnel. La première dimension correspond au choix d'un opérateur de destruction $d \in D$. La deuxième dimension correspond au choix d'une opérateur de réparation $r \in R$. La troisième dimension correspond au choix d'un degré de destruction de la solution par l'opérateur de destruction. On notera par $\mathcal{P}$ l'ensemble des degrés de destruction. Une action est donc représentée par un vecteur de taille 3. On notera $A = A^1 \cup A^2 \cup A^3$ où A^1, A^2 et A^3 représentent respectivement les espaces d'actions sur le choix des opérateurs de destruction, le choix des opérateurs de réparation et le choix des degrés de destruction. On notera une action $\mathbf{a}_t \in A$ par $\mathbf{a}_t = (a_t^1, a_t^2, a_t^3)$, où $a_t^i \in A_i \quad i \in \{1, 2, 3\}$. Avec ces notations, l'espace A^1 (respectivement A^2) est un ensemble d'indices allant de 1 au nombre d'opérateurs de destruction (respectivement de construction) utilisés. L'espace A_3 est un ensemble d'indices allant de 1 au nombre de degrés de destruction utilisés. A partir de l'action $\mathbf{a}_t$ on récupère les opérateurs et le degré $deg \in \mathcal{P}$ correspondants, où $\mathcal{P}$ est l'ensemble des dégrés de destruction utilisés. La taille du vecteur du vecteur $\mathbf{a}_t$ est égal à $|D| \times |R| \times |\mathcal{P}|$. En outre, le nombre d'actions $\mathbf{a}_t$ différentes est de $|A| = |D| \cdot |R| \cdot |\mathcal{P}|$. Pour limiter la taille de $|A|$ et donc la complexité du modèle, nous avons limité $\mathcal{P}$ aux deux valeurs $\{1, 20\%\}$. Pour $deg = 1$ (respectivement 20%) on détruit une visite (respectivement 20% des visites) de la solution. Lorsque nous faisons rouler *ALNS*, un degré de destruction est choisi aléatoirement parmi $\mathcal{P}$ à chaque itération.

Représenter l'action par une seule valeur a_t aurait aussi été possible. Dans ce cas a_t est un vecteur unidimensionnel de taille $|D| \cdot |R| \cdot |\mathcal{P}|$. Nous avons choisi la représentation multidimensionnelle pour diminuer la taille du vecteur d'action et afin de diminuer la complexité du modèle.

3.6.5 Récompenses

Une récompense est une valeur quantitative qui exprime la performance du modèle quant à la réalisation d'objectifs prédéfinis. Les récompenses $\mathbf{r}_t$ sont calculées selon la fonction de récompense R par $\mathbf{r}_t = R(s_t, a_t, s_{t+1})$. Nous avons implémenté cette fonction en nous basant sur l'étude [31]. Les auteurs proposent une fonction de récompense dont la formule est donnée par l'équation (3.12).

$$r_t = \left[\frac{f(x_t) - f(x_t^*)}{f(x_t^*)} \right] \tag{3.12}$$

Nous avons choisi cette fonction de récompense car en cherchant à la maximiser, le modèle

va faire diminuer le coût $\mathbf{f}(x_t^*)$ par la même occasion. Dans [31], la fonction de coût f est à valeurs réelles. Par conséquent la récompense r_t est un scalaire. Cependant, la fonction de coût que nous utilisons est un vecteur multidimensionnel $\mathbf{f} \in \mathbb{R}^3$. En utilisant la fonction de récompense donnée par l'équation (3.12), la récompense appartient à $\mathbb{R}^3$. Lorsque les récompenses sont des vecteurs, le problème est un *MOMDP* [37]. Ces problèmes sont plus complexes et nous avons cherché à rester dans le cas des récompenses à valeurs réelles. Pour cela nous avons introduit un vecteur de poids $\boldsymbol{a}$ et une fonction de scalarisation $u_{\boldsymbol{a}}$ dont les formules sont données par les équations (3.13) et (3.14).

$$\boldsymbol{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_K \end{pmatrix} \in \mathbb{R}_+^{\mathbb{K}} \text{ et } \sum_{k=1}^{K} a_k = 1 \tag{3.13}$$

$$\begin{aligned} u_{\boldsymbol{a}} : \mathbb{R}^K &\longrightarrow \mathbb{R} \\ \mathbf{y} &\longmapsto \boldsymbol{a}^T \cdot \mathbf{y} \end{aligned} \tag{3.14}$$

Nous appliquons $u_{\boldsymbol{a}}$ à $\mathbf{f}$ pour obtenir un scalaire selon l'équation (3.15).

$$f = u_{\boldsymbol{a}}(\mathbf{f}) \tag{3.15}$$

Notre fonction de récompense s'écrit selon l'équation (3.16).

$$r_t(x_t, x_t^*) = \left[\frac{u_{\boldsymbol{a}}(\mathbf{f}(x_t)) - u_{\boldsymbol{a}}(\mathbf{f}(x_t^*))}{u_{\boldsymbol{a}}(\mathbf{f}(x_t^*))} \right] \tag{3.16}$$

Nous avons déterminé les valeurs du vecteur $\boldsymbol{a}$ de telle sorte que la fonction de récompense vérifie l'équation (3.17).

$$r_t(x_t, x_t^*) \leq r_t(x_t', x_t^*) \implies x_t \prec x_t' \tag{3.17}$$

Ceci est important pour la compréhension de l'environnement par le modèle. En effet, celui-ci doit pouvoir apprendre que si une solution x_t' a une récompense plus élevée que la solution x_t alors cela implique que x_t' est meilleure que x_t. Le modèle va donc chercher des solutions de meilleur qualité. En étudiant les plages de valeurs des coûts sur les Visites rejetées, la Fidelité et la Distance, nous avons déterminé que la Fidelité ne dépassait jamais 10^3 et que la Distance ne dépassait jamais 10^5. Pour le scénario $S1$, on a choisi comme vecteur de

poids : $\boldsymbol{a} = (1, 10^{-3}, 10^{-8})$. Et pour le scénario $S2$, on a choisi comme vecteur de poids : $\boldsymbol{a} = (1, 10^{-5}, 10^{-8})$. De cette façon, l'équation (3.17) est toujours vérifiée dans chacun des scénarios.

3.7 Entraînement des modèles

Nous avons employé les algorithmes *Proximal Policy Optimization* (*PPO*) [39] et *Soft Actor-Critic* (*SAC*) [40] pour entraîner les modèles *ALNS-PPO* et *ALNS-SAC*. Ces deux algorithmes ont été employés dans leur version discrète, c'est-à-dire pour les cas où l'espace des actions est discret. De plus, *PPO* et *SAC* comportent la même architecture en Acteur-Critique 3.8.

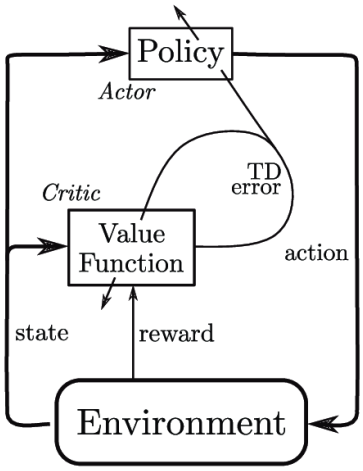


FIGURE 3.8 Architecture du modèle de l'Acteur-Critique. Illustration tirée de [1]

L'architecture en acteur-critique est une architecture populaire de modèle de *RL* qui fusionne les stratégies qui sont basées sur les fonctions d'évaluation et celles qui sont basées sur les politiques. Les modèles en acteur-critique sont composés d'un acteur et d'un critique. L'acteur sélectionne les actions suivant sa politique π_θ et le critique juge la qualité du choix de l'acteur selon sa fonction d'évaluation V. Celle-ci prend en entrée l'action a_t sélectionnée par l'acteur et retourne une valeur qui cherche à rendre compte de la qualité de l'action. La valeur retournée peut par exemple être une estimation du retour dévalué obtenu en choisissant l'action a_t dans l'état s_t. Le rôle du critique peut être donc vu comme un retour sur les choix de l'acteur.

Pour les modèles de type acteur-critique, l'acteur et le critique sont mis à jour selon des

fonctions de perte (ou *loss*) différentes. Généralement, l'acteur et le critique ne partagent pas le même réseau de neurones. Ce sera notre choix pour cette étude.

Nous explicitons ci-dessous les versions employées de *PPO* et *SAC* pour l'entraînement des modèles *ALNS-PPO* et *ALNS-SAC*. On utilise l'implémentation de la librarie CleanRL [41] et de Spinning Up d'OpenAI [42] pour implémenter les algorithmes dans leur version multi-discrète.

Pour *ALNS-PPO* et *ALNS-SAC*, avant la phase d'entraînement, nous avons inclus une phase d'échauffement qui consiste à faire rouler *ALNS* un certain nombre de fois et de récupérer les observations de chaque *run*. A la fin de cette phase, on calcule les statistiques de normalisation des observations. Pour chaque observation, soit on ne la normalise pas, soit on la normalise et auquel cas on utilise une normalisation par min-max ou par standardisation. L'approche que nous suivons est de normaliser par min max les variables dont on connaît les bornes et de normaliser par standardisation les autres. Les variables que nous ne normalisons pas sont en fait les variables binaires de l'observation. On a estimé qu'il n'était pas nécessaire de les normaliser. Le tableau 3.5 donne le type de normalisation employé par variable de l'observation.

Tableau 3.5 Type de normalisation employée par caractéristique de l'observation

Caractéristiques	Type de normalisation
n_{P}	min - max
n_{A}	min - max
$f(x_t)$	standard
$f(x'_t)$	standard
$f(x^*_t)$	standard
Δf_t	standard
Δf^*_t	standard
$1_{\{x'_t \succ x_t\}}$	pas de normalisation
$1_{\{x'_t \succ x^*_t\}}$	pas de normalisation
$1_{\{x'_t \prec x_t\,,\,\mathrm{acc}\}}$	pas de normalisation
$1_{\{x'_t \prec x_t\,,\,\overline{\mathrm{acc}}\}}$	pas de normalisation
t	min - max
$\boldsymbol{a}_{t-1}$	pas de normalisation
T	min - max
n_{seen}	min - max
$n_{\mathrm{last\_best}}$	min - max

L'algorithme de la phase d'échauffement est fourni dans l'annexe F. Pendant la phase d'échauffement comme pendant la phase d'entraînement, les trajectoires sont collectée selon l'algorithme fourni dans l'annexe G. Cet algorithme consiste collecter itérativement tous les pas

de la trajectoire jusqu'à ce que la taille de la trajectoire soit atteinte. L'algorithme qui collecte un pas de trajectoire est founi dans l'annexe H. Pendant la phase d'entraînement, les observations sont normalisées selon l'algorithme fourni dans l'annexe I.

Fonctionnement du modèle *ALNS-PPO*

Le modèle *ALNS-PPO* utilise l'algorithme *PPO*. *PPO* est un algorithme populaire introduit par [39]. Cet algorithme est *on-policy*, c'est-à-dire que la politique entraînée est celle qui est utilisée pour collecter les expériences. *PPO* appartient à la classe des *Policy Based Methods* qui sont les méthodes qui cherchent à mettre à jour directement la politique du modèle. Il est une variante de l'algorithme *REINFORCE* dans le fait qu'il ajoute une stabilité sur la mise à jour de la politique.

Le pseudo-code de *ALNS-PPO* est fourni dans la figure 3.9. La version de l'algorithme *PPO* intégré à *ALNS-PPO* inclut une régularisation de l'entropie de la politique dans la *loss* sur l'acteur ainsi qu'un *clipping* sur la *loss* du critique.

Algorithm 4 *ALNS-PPO*

Require: Set d'instances U (input)
Ensure: Politique π_θ (output)
function *ALNS-PPO*(u)
    Initialiser l'acteur π_θ, le critique ψ_ϕ
    Initialiser l'ensemble des trajectoires d'échauffement $\mathcal{D}^E = \{\}$
    Initialiser les moyennes $\mathbf{M}$ et écarts-types $\mathbf{\Sigma}$
    $\theta_{old} \leftarrow \theta$, $\phi_{old} \leftarrow \phi$

    for tour d'échauffement $= 1, 2, \ldots, N^E$ **do**
        for all instance u dans U **do**
            $\tau \leftarrow$ CollecterTrajectoire($u, \pi_\theta, \psi_\phi, \mathbf{M}, \mathbf{\Sigma}$)
            Ranger τ dans $\mathcal{D}^E$
        end for
    end for
    $\mathbf{M}, \mathbf{\Sigma} \leftarrow$ CalculerStats($\mathcal{D}^E$) ▷ Calculer les moyennes et écarts-types sur $\mathcal{D}^E$

    for each instance u de U et **for each** pas de mise à jour **do**
        for mise à jour $= 1, 2, \ldots, N$ **do**
            $\tau \leftarrow$ CollecterTrajectoire($u, \pi_{\theta_{old}}, \psi_{\phi_{old}}, \mathbf{M}, \mathbf{\Sigma}$)
            $R^\lambda \leftarrow$ CalculerLambdaReturn($\tau, \psi_{\phi_{old}}$)
            $\hat{A} \leftarrow$ CalculerGAELambda($\tau, \psi_{\phi_{old}}, R^\lambda_t$)
            for époque $= 1, 2, \ldots, E$ **do**
                $\pi_{\theta_{old}} \leftarrow \pi_\theta$
                $\psi_{\phi_{old}} \leftarrow \psi_\phi$
                for all mini-batch $\boldsymbol{b}_\tau$ de la trajectoire τ **do**
                    $L^\pi(\theta) \leftarrow$ CalculerActorLoss($\pi_\theta, \boldsymbol{b}_\tau, \hat{A}$)
                    $L^{\mathrm{V}}(\phi) \leftarrow$ CalculerCriticLoss($\psi_\phi, \boldsymbol{b}_\tau, R^\lambda$)
                    $\theta \leftarrow \theta - \alpha \nabla_\theta L^\pi(\theta)$
                    $\phi \leftarrow \phi - \beta \nabla_\phi L^{\mathrm{V}}(\phi)$
                end for
            end for
        end for
    end for
    return π_θ
end function

FIGURE 3.9 Pseudocode du modèle *ALNS-PPO*

Dans *PPO*, la fonction d'évaluation V du critique calcule une estimation des *avantages* de l'acteur. Des avantages positifs, respectivement négatifs, représentent une sous-estimation,

respectivement une surestimation, de retour dévalué par le critique. Dans notre cas, les avantages sont calculés par la méthode de *GAE* lambda [18] qui utilise le retour dévalué lambda (*lambda return*). Le calcul du retour dévalué est donné dans l'annexe J. La méthode *GAE* est donnée par dans l'annexe *K*.

La *loss* sur l'acteur est donnée dans l'annexe L. Cette *loss* est une somme pondérée entre la CLIP *loss* sur la politique π_θ et la *loss* sur l'entropie. L'innovation de l'algorithme *PPO* se situe dans le calcul de la CLIP *loss* qui fait intervenir comme nouvelle variable le rapport de probabilité d'obtenir a_t sachant s_t. Ce rapport permet de stabiliser la variation de la politique π_θ en restreignant l'amplitude des mises-à-jour de π_θ grâce au paramètre ϵ.

La *loss* du critique est donné dans l'annexe M. Cette *loss* est le maximum entre l'erreur quadratique entre la fonction d'évaluation du critique et le retour dévalué d'une ancienne trajectoire et cette même erreur mais clippée par le paramère ϵ.

Fonctionnement du modèle *ALNS-SAC*

Le modèle *ALNS-SAC* utilise l'algorithme *SAC*. *SAC* est un algorithme populaire en *RL* qui est *off-policy*, c'est-à-dire qu'il entraîne une politique à partir d'expériences qui peuvent être produites par des politiques plus anciennes via son *replay buffer*. La spécificité de *SAC* est qu'il cherche à maximiser l'entropie H_t sur le choix des actions pour favoriser une exploration plus diversifiée de l'environnement. L'entropie désigne ici l'entropie de Shannon dont la formule est donnée dans l'équation (3.18).

$$H_t = -\sum_{a \in \mathcal{A}} \pi(a|s_t) \log(\pi(a|s_t)) \tag{3.18}$$

Chercher à maximiser l'entropie H_t revient à maximiser la l'incertitude sur le choix de l'action a_t. Cela incite par conséquent le modèle à diversifier son exploration.

Le pseudo-code de *ALNS-SAC* est fourni dans la figure 3.10.

Algorithm 5 *ALNS-SAC*

Require: Set d'instances U (input)
Ensure: Politique π_θ (output)

function ALNS-SAC(U)
  Initialiser l'acteur π_θ, le critique Q_{ϕ_1,ϕ_2}, le *target* critique $Q_{\phi'_1,\phi'_2}$
  Initialiser les paramètres du *target* $\phi'_i \leftarrow \phi_i$ pour $i \in \{1, 2\}$
  Initialiser le *replay buffer* $\mathcal{D}$, les trajectoires d'échauffement $\mathcal{D}^E$
  Initialiser les moyennes $\mathbf{M}$ et écarts-types $\mathbf{\Sigma}$, le pas $\mathcal{P}$ et la température α

  for tour d'échauffement $= 1, 2, \ldots, N^E$ **do**
    for all instance u dans U **do**
      $\tau \leftarrow$ CollecterTrajectoire$(u, \pi_\theta, \psi_\phi, \mathbf{M}, \mathbf{\Sigma})$
      Ranger τ dans $\mathcal{D}^E$
    end for
  end for
  $\mathbf{M}, \mathbf{\Sigma} \leftarrow$ CalculerStats$(\mathcal{D}^E)$ ▷ Calculer les moyennes et écarts-types sur $\mathcal{D}^E$

  for each instance u de U et **for each** pas de mise à jour **do**
    for $t = 0, 1, \ldots, K$ **do**
      $\mathcal{P} \leftarrow$ CollecterUnPas$(\mathcal{P}, \pi_\theta, \psi, \mathbf{M}, \mathbf{\Sigma})$
      Ranger $\mathcal{P}$ dans $\mathcal{D}$
      if t mod $N_{net} == 0$ **then** ▷ toutes les N_{net} itérations
        Echantillonner un batch $\boldsymbol{b}_\mathcal{D}$ du *replay buffer* $\mathcal{D}$
        $L^\pi(\theta) \leftarrow$ CalculerActorLoss$(\boldsymbol{b}_\mathcal{D})$

$$\theta \leftarrow \theta - \eta_\pi \nabla_\theta L^\pi(\theta)$$

        $L^Q(\phi_1, \phi_2) \leftarrow$ CalculerQLoss$(\boldsymbol{b}_\mathcal{D})$

$$\phi_i \leftarrow \eta_Q \nabla_{\phi_i} L^Q(\phi_1, \phi_2) \quad \text{pour } i \in \{1, 2\}$$

        $L(\alpha) \leftarrow$ CalculerAlphaLoss$(\boldsymbol{b}_\mathcal{D})$

$$\alpha \leftarrow \alpha - \eta_\alpha \nabla_\alpha L(\alpha)$$

      else if t mod $N_{net}^{target} == 0$ **then** ▷ toutes les N_{net}^{target} itérations
        on met à jour les paramètres du *target-network*

$$\phi'_i \leftarrow \tau\phi_i + (1 - \tau)\phi'_i \quad \text{pour } i \in \{1, 2\}$$

      end if
    end for
  end for
  return π_θ
end function

FIGURE 3.10 Pseudocode du modèle *ALNS-PPO*

L'algorithme de calcul de la *loss* sur l'acteur est fourni dans l'annexe N. Diminuer cette *loss* revient à paramétrer le réseau de l'acteur de telle sorte qu'on maximise l'espérance sur la valeur des fonctions d'évaluation du critique et de de la valeur de l'entropie.

L'algorithme de calcul de la *loss* sur le critique est fourni dans l'annexe O. Diminuer cette *loss* revient à paramétrer le réseau du critique de telle sorte qu'on minimise l'écart entre la valeur réelle du retour dévalué et son estimation par le critique.

L'algorithme de calcul de la *loss* sur l'entropie du modèle est fourni dans l'annexe P. Cette *loss* fait intervenir un paramètre α et une valeur d'entropie à atteindre $\bar{H}$. Diminuer la *loss* sur l'entropie revient à paramétrer la valeur de α de telle sorte que l'entropie du modèle H_t tende vers la valeur de $\bar{H}$.

CHAPITRE 4 RÉSULTATS

Dans le chapitre sur la méthodologie nous avons développé une méthode qui intégre du *RL* à la partie adaptative de sélection des opérateurs de *ALNS*. Cette méthode met en jeu les modèles *ALNS-PPO* et *ALNS-SAC*, deux algorithmes populaires de *RL* et dont les approches d'apprentissage sont *on-policy* et *off-policy* respectivement. Ce chapitre se conscare à la génération d'un ensemble de données et à la comparaison des performances des deux modèles de *RL* à *ALNS* sur l'ensemble de données.

4.1 Description des données

Afin de pouvoir étudier la performance des modèles *ALNS-PPO* et *ALNS-SAC* il faut avoir accès à un ensemble de données d'entraînement suffisamment grand. Dans cette étude l'ensemble d'instances utilisé a été généré artificiellement, en se basant sur un ensemble d'instances réelles. Nous présentons dans cette section comment se définit une donnée puis par quelle méthode l'ensemble de données a été généré.

L'algorithme *ALNS* prend en entrée une donnée appelée instance. Une instance se caractérise par ses données sur les patients, les aides-soignants, les valeurs de fidélité entre patients et aides-soignants, les matrices de distance et durée par trajet et sur les visites préprogrammées (ou fixes). Ces données sont présentées ci-dessous.

Caractéristiques d'un patient

Les caractéristiques des patients sont données dans le tableau 4.1. Un patient est caractérisé par un identifiant unique (*Id*), des coordonnées géographiques (*Localisation*), des expertises requises par l'aide-soignant pour le soigner (*Expertises*), la durée d'une visite *Durée*, un nombre de visites (*Visites*) et une fenêtre de temps de disponibilité (*Disponibilité*).

Tableau 4.1 Description d'un patient

Nom	Description
Id	Identifiant.
Localisation	Coordonnées géographiques.
Expertises	Expertises requises pour sa visite.
Durée	Durée d'une visite.
Visites	Nombre de visites nécessaires pour soigner le patient.
Disponibilité	Fenêtre de temps de disponibilité du patient pour la visite.

Caractéristiques d'un aide-soignant

Les caractéristiques de chaque aide-soignant sont données dans le tableau 4.2. Un aide-soignant est caractérisé par un identifiant unique (*Id*), des coordonnées géographiques (*Localisation*), des expertises (*Expertises*) et une fenêtre de temps de disponibilité journalière (*Disponibilités*).

Tableau 4.2 Description d'un aide-soignant

Nom	Description
Id	Identifiant.
Localisation	Coordonnées géographiques.
Expertises	Expertises de l'aide-soignant.
Disponibilités	Fenêtres de temps de disponibilité journalière.

Caractéristiques de la fidélité patient à aide-soignant

Les caractéristiques des fidélités entre chaque patient et aide-soignant sont données dans le tableau 4.3. La fidélité se définit comme la qualité de l'intéraction entre le patient et son aide-soignant. Cette grandeur permet de rendre compte de la continuité des soins d'un aide-soignant à un même patient. Plus sa valeur est grande, plus la qualité de l'intéraction entre le patient et l'aide-soignant est grande. Pour chaque fidélité les identifiants du patient et de l'aide-soignant sont donnés respectivement par *Id Patient* et *Id Aide-Soignant* et la valeur de la fidélité est donnée par *Valeur*.

Tableau 4.3 Description de la fidélité patient à aide-soignant

Nom	Description
Id Patient	Identifiant du patient.
Id Aide-Soignant	Identifiant de l'aide-soignant.
Valeur	Valeur de la fidélité.

Caractéristiques d'un trajet

La matrice des distances et durées par trajet décrit les caractéristiques du trajet de chaque paire de couple (patient, aide-soignant) et (aide-soignant, patient) et de chaque paire de couple (patient_1, patient_2) et (patient_2, patient_1). Ces caractéristiques sont données dans le tableau 4.4. Un trajet est caractérisé par une distance (*Distance*) et d'une durée qui correspond au temps passé à parcourir le trajet (*Temps*).

TABLEAU 4.4 Description d'un trajet entre deux patients ou entre un patient et un aide-soignant

Nom	Description
Distance	Distance du trajet.
Temps	Temps à passer pour parcourir le trajet.

Caractéristiques d'une visite préprogrammée

Les caractéristiques des visites préprogrammées sont données dans le tableau 4.5. Une visite préprogrammée se caractérise par l'identifiant d'un patient (*Id Patient*), l'identifiant d'un aide-soignant (*Id Aide-Soignant*), la date de la visite (*Date*) et la durée de la visite (*Durée*).

TABLEAU 4.5 Description d'une visite préprogrammée

Nom	Description
Id Patient	Identifiant du patient.
Id Aide-Soignant	Identifiant de l'aide-soignant.
Date	Moment (date et heure) auquel commence le service.
Durée	Durée du service.

4.1.1 Génération d'instances artificielles

Nous avons généré un ensemble D' à partir d'un ensemble d'instances D fourni par notre partenaire Alayacare. Nous décrivons ci-dessous les différentes étapes de la création de D'.

Explication de la méthode de génération d'instances

La méthode de génération d'instance est la suivante. D'abord on estime des distributions sur les valeurs associées à certaines caractéristiques. Puis de nouvelles instances sont générées à partir de ces distributions.

Les principales étapes de la génération de l'ensemble D' :

- Echantillonner un couple (n_P, n_A) depuis un ensemble $\mathcal{N}_{(P,A)}$.
- Créer une zone géographique dans laquelle générer les coordonnées (lat, lon) des patients et aides-soignants.
- Calculer les temps de trajet entre les individus (excepté entre deux aides-soignants).
- Attribuer des expertises aux patients et aux aides-soignants.
- Attribuer des valeurs de fidélité entre les patients et les aides-soignants.
- Attribuer des fenêtres de temps de disponibilité aux patients.
- Attribuer des horaires aux aides-soignants.

— Calculer la durée des visites des patients.
— Déterminer le nombre de visites des patients et déterminer l'horizon temporel de résolution T.

Détermination de l'ensemble $\mathcal{N}_{(P,A)}$

Pour déterminer l'ensemble $\mathcal{N}_{(P,A)}$, nous avons analysé la dispersion des couples du nombres de patients et d'aides-soignants (n_P, n_A) de l'ensemble D. La figure (4.1) illustre cette dispersion ainsi qu'une enveloppe convexe en rouge comprenant ces points. Nous avons choisi $\mathcal{N}_{(P,A)}$ comme étant cette enveloppe convexe. Celle-ci se définit par les points (n_P, n_A) : $(2, 1)$, $(50, 1)$, $(80, 10)$, $(80, 14)$, $(2, 14)$.

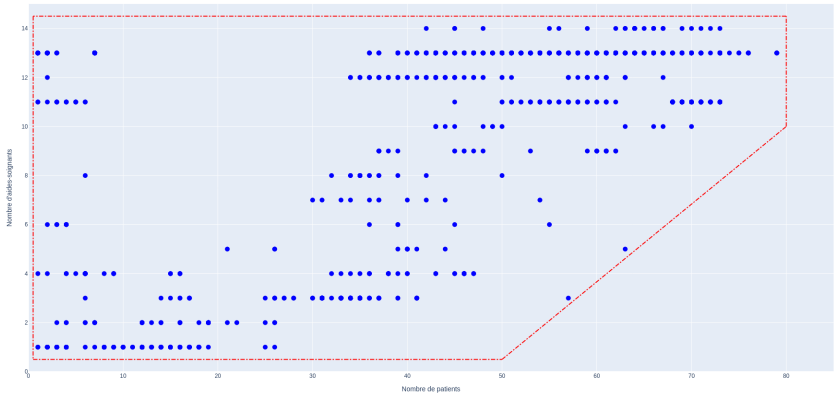


FIGURE 4.1 Dispersion des couples du nombre de patients et d'aides-soignants de l'ensemble D

Générer les localisations

Pour générer des localisations il faut d'abord générer une zone spatiale qui les contient. On veut en outre que la taille de la zone soit représentative de l'ensemble D. Pour cela, on cherche à déterminer sur D la dispersion spatiale des coordonnées des patients et aides-soignants. On cherche à définir des grandeurs qui rendent compte de cette dispersion spatiale. Nous définissons ainsi les variables d_{lat} et d_{lon} dont les formules sont données par l'équation 4.1.

$$\begin{aligned} d_{lat} &= d^{geo}(min_i(lat), max_i(lat)), \\ d_{lon} &= d^{geo}(min_i(lon), max_i(lon)) \\ &\text{pour tout couple de coordonnées } (lat, lon) \text{ d'une instance } i \end{aligned} \tag{4.1}$$

La fonction utilisée pour calculer ces distances est la fonction de calcul de la distance géodésique de la librairie Python *geopy.distance*. Les calculs de cette fonction sont basés sur les formules de Vincenty. d_{lat} et d_{lon} sont respectivement les côtés en latitude et en longitude du rectangle de surface minimale englobant toutes les coordonnées d'une instance donnée.

Pour générer une zone spatiale, on génère un couple (d_{lat}, d_{lon}) selon la distribution jointe sur les valeurs de (d_{lat}, d_{lon}) de l'ensemble D par estimation par noyau (KDE). La zone spatiale qu'on note $R_{d_{lat},d_{lon}}$ est construite comme un rectangle de côté d_{lat} en latitude et de côté d_{lon} en longitude. La figure (4.2) donne la dispersion des valeurs (d_{lat}, d_{lon}) sur D. On peut voir que les couples de distances (d_{lat}, d_{lon}) sont répartis entre quelques kilomètres et une vingtaine kilomètres, ce qui correspond à des distances caractéristiques de zones urbaines ou de zones périurbaines.

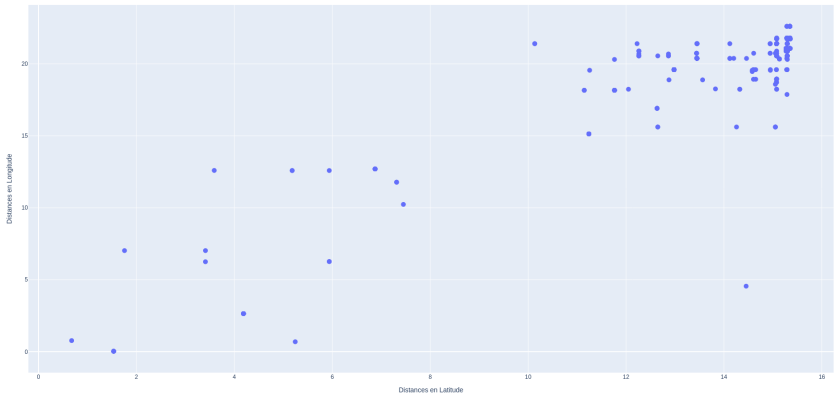


FIGURE 4.2 Dispersion des distances caractéristiques d_{lat} et d_{lon} de l'ensemble D (km)

On génère en outre les localisations des patients et des aides-soignants dans $R_{d_{lat},d_{lon}}$ selon 2 modes. Dans le premier mode on génère aléatoirement dans $R_{d_{lat},d_{lon}}$. Dans le deuxième mode on génère des clusters en disque et dont les centroïdes sont générés aléatoirement dans $R_{d_{lat},d_{lon}}$. On choisit c dans $\{3, 5\}$. On calcule les rayons des clusters selon l'équation (4.2).

$$\text{r} = \frac{d_{lat} + d_{lon}}{2c} \text{ avec } r \text{ le rayon et } c \text{ le nombre de clusters} \tag{4.2}$$

On utilise cette formule pour plusieurs raisons :

— on s'assure que les clusters sont de taille appropriée par rapport à la surface totale de $R_{d_{lat},d_{lon}}$. Cela permet aussi de les répartir uniformément dans la $R_{d_{lat},d_{lon}}$

— avec cette approche on simule différentes densités de population au sein de $R_{d_{lat},d_{lon}}$.

La densité de population augmente quand la taille des clusters diminue

— les clusters ne se chevauchent pas de manière excessive

— la taille des clusters est proportionnelle à la zone $R_{d_{lat},d_{lon}}$

avec d^{geo} étant la fonction de calcul de la distance géodésique de la librairie Python *geopy.distance*.

Les figures 4.3, 4.4 et 4.5 fournissent un exemple de génération des localisations de 100 individus selon les modes *Random*, $Cluster-3$ et $Cluster-5$ respectivement.

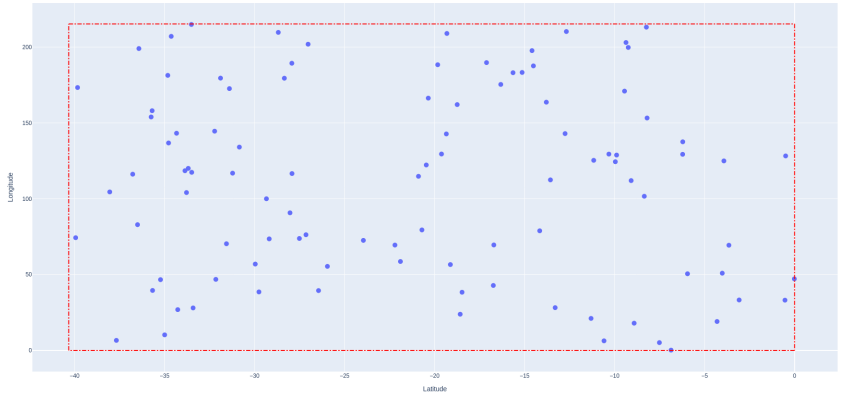


FIGURE 4.3 Exemple d'une génération des localisations selon le mode *Random*

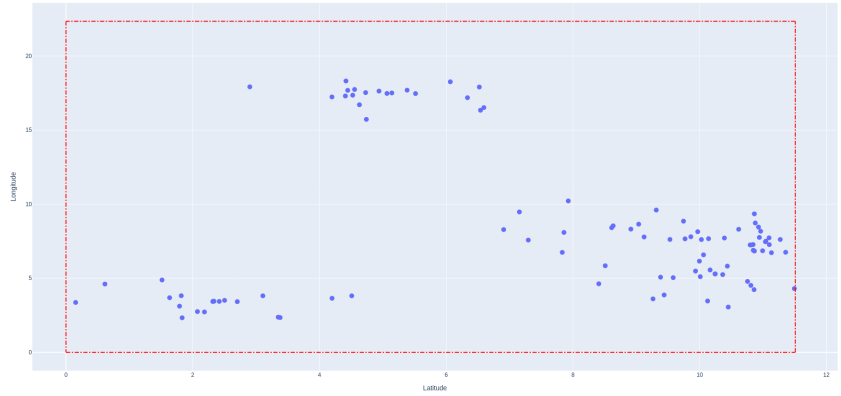


FIGURE 4.4 Exemple d'une génération des localisations selon le mode $Cluster-3$

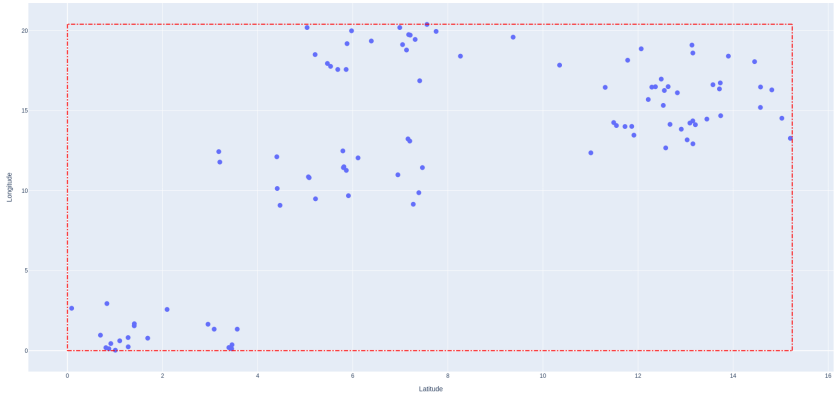


FIGURE 4.5 Exemple d'une génération des localisations selon le mode $Cluster - 5$

Calculer les temps de trajet

La figure 4.6 donne la dispersion des temps de trajet en fonction de la distance de ces derniers. On applique une régression linéaire segmentée de 3 segments à cette dispersion.

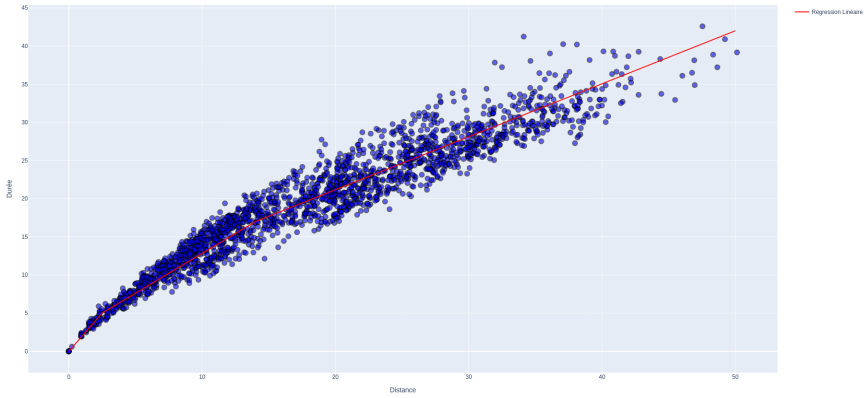


FIGURE 4.6 Régression linéaire selon 3 segments du temps en fonction de la distance des trajets

Le nombre 3 a été choisi pour faire correspondre au mieux la régression aux valeurs. Les coefficients de la régression sont donnés par les équations (4.3) et (4.4).

$$\mathcal{D}_{t|d} = \begin{cases} b_1 + a_1 d, & \text{pour } d \leq d_1 \\ b_2 + a_2 d, & \text{pour } d_1 < d \leq d_2 \\ b_3 + a_3 d, & \text{pour } d > d_2 \end{cases} \tag{4.3}$$

avec

$$\begin{aligned}(a_1, b_1) &= (2.0, 0.0)\\ (a_2, b_2) &= (1.0, 2.4)\\ (a_3, b_3) &= (0.7, 7.3)\end{aligned} \tag{4.4}$$

Attribution des valeurs de fidélité

Dans cette section nous cherchons à déterminer une estimation $\mathcal{D}^f_{n_P,n_A}$ de la distribution réelle des valeurs de fidélité entre patients et aides-soignants en fonction du couple (n_P, n_A). La figure 4.7 donne les fonctions de densité des estimations par noyau des distributions des valeurs de fidélité par instance de D. On a réparti les estimations en 3 clusters.

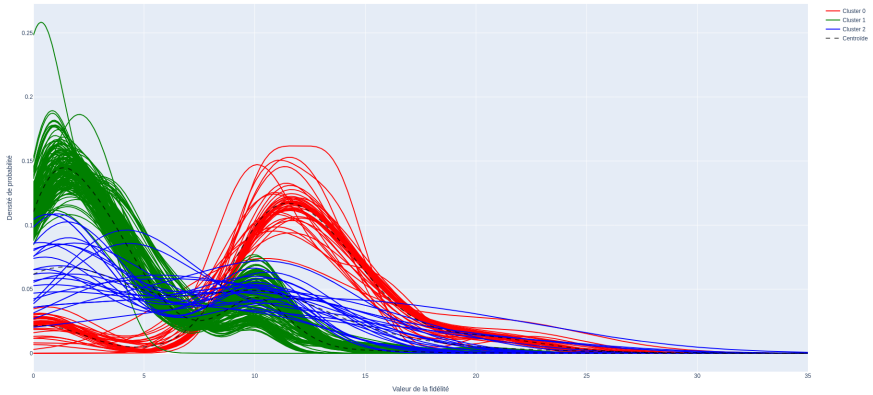


FIGURE 4.7 Densité de probabilité des estimations par noyau des distributions

La figure 4.8 représente la dispersion des 3 clusters selon (n_P, n_A).

FIGURE 4.8 Dispersion par cluster des estimations par noyau des distributions des valeurs sur la fidélité sur D

On note par $(\hat{\boldsymbol{c}}_i, |\boldsymbol{c}_i|)$ le couple (centroïde, taille du cluster i) pour i variant dans $\{r \text{ (rouge)}, v \text{ (vert)}, b \text{ (bleu)}\}$. Les tailles des clusters sont : $|\boldsymbol{c}_r| = 273$, $|\boldsymbol{c}_v| = 192$ et $|\boldsymbol{c}_b| = 35$. A partir des valeurs données dans les figures 4.7 et 4.8, nous avons déterminé à quel centroïde associer $\mathcal{D}^f_{n_P,n_A}$ à partir de probabilités données ci-dessous. On a calculé ces probabilités en considérant de manière simplifiée que pour la zone correspondante à $n_P \leq 10$ et $n_A \leq 4$, la probabilité d'appartenir au cluster bleu est de $\frac{1}{2}$ et qu'en dehors de cette zone, elle était nulle. Si les probabilités donnent que $\mathcal{D}^f_{n_P,n_A}$ n'appartient pas au cluster bleu, la probabilité d'appartenir à l'un ou l'autre cluster est calculée selon leur taille relative. Les probabilités sont données dans l'équation (4.5).

$$
\begin{aligned}
n_P \leq 10 \text{ et } n_A \leq 4 \implies & \begin{cases} p(\mathcal{D}^f_{n_P,n_A} = \hat{\boldsymbol{c}}_b) = \frac{1}{2} \\ p(\mathcal{D}^f_{n_P,n_A} = \hat{\boldsymbol{c}}_r) = \frac{1}{2}\frac{|\boldsymbol{c}_r|}{|\boldsymbol{c}_r|+|\boldsymbol{c}_v|} = 0.29 \\ p(\mathcal{D}^f_{n_P,n_A} = \hat{\boldsymbol{c}}_v) = \frac{1}{2}\frac{|\boldsymbol{c}_v|}{|\boldsymbol{c}_r|+|\boldsymbol{c}_v|} = 0.21 \end{cases} \\
n_P > 10 \text{ ou } n_A > 4 \implies & \begin{cases} p(\mathcal{D}^f_{n_P,n_A} = \hat{\boldsymbol{c}}_b) = 0 \\ p(\mathcal{D}^f_{n_P,n_A} = \hat{\boldsymbol{c}}_r) = \frac{|\boldsymbol{c}_r|}{|\boldsymbol{c}_r|+|\boldsymbol{c}_v|} = 0.59 \\ p(\mathcal{D}^f_{n_P,n_A} = \hat{\boldsymbol{c}}_v) = \frac{|\boldsymbol{c}_v|}{|\boldsymbol{c}_r|+|\boldsymbol{c}_v|} = 0.41 \end{cases}
\end{aligned}
\tag{4.5}
$$

Attribution des expertises

Dans les instances d'Alayacare, les nombre d'expertises différentes mises en jeu est de 3. Pour les aides-soignants, on génère les expertises des aides-soignants en tirant aléatoirement

k expertises parmi les 3, où k est un nombre aléatoire entre 0 et 3 inclus. Pour les patients, on génère les expertises selon 3 scénarios. A chaque scénario S_i^E, $i \in \{1, 2, 3\}$ on associe le nombre d'expertises requises maximales égal à i. On tire ensuite aléatoirement k expertises parmi les 3, où k est un nombre aléatoire entre 0 et i inclus. Nous utilisons ces distributions de probabilité pour échantillonner les expertises par patients et par aide-soignant.

Détermination des fenêtres de disponibilité des patients

On note les fenêtres de disponibilité des patients par $[t_0, t_F]$. Pour la plage de valeur de t_0, on prendra la même que celle de l'ensemble D, à savoir entre 6h et 21h. On génère donc aléatoirement t_0 dans cette plage. On écrit la valeur de t_F comme $t_0 + \Delta t$. La figure 4.9 donne la distribution de l'estimation par noyau de Δt sur D. On utilisera cette distribution pour générer Δt. Par ailleurs, on borne la valeur de t_F à 24h, qui est la valeur maximale de t_f sur D, soit $t_F = \min(t_0 + \Delta t, 24h)$.

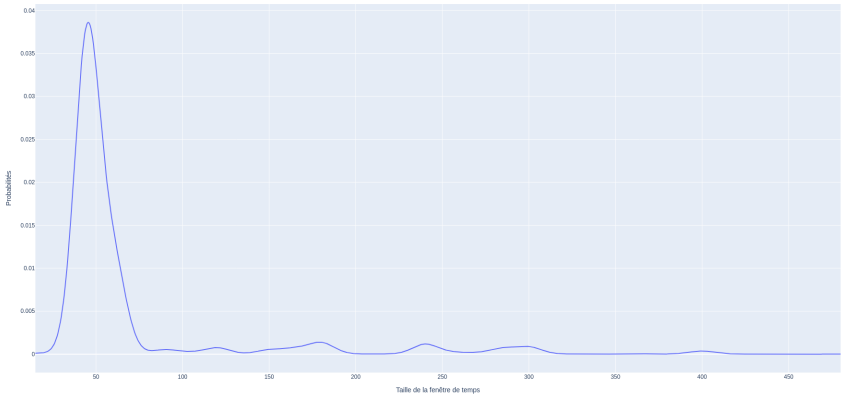


FIGURE 4.9 Distribution de probabilité de l'estimation par noyau de Δt sur D

Détermination des horaires des aides-soignants

Pour les aides-soignants, on considère que leur fenêtre de disponibilité journalière est de 6 heures à 24 heures. On a choisi ces horaires car ils correspondent aux temps minimal et maximal des plages horaires de t_0 et t_F des fenêtres de temps de disponibilité des patients.

Détermination de la durée des visites des patients

La figure 4.10 donne les probabilités d'occurence par durée de service pour l'ensemble D. Nous utilisons ces distributions de probabilité pour générer les durées des visites.

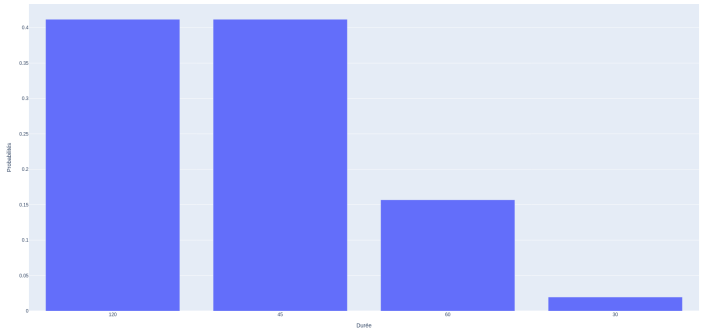


FIGURE 4.10 Probabilités d'occurence par durée de service des patients sur l'ensemble D.

Détermination des visites et de l'horizon temporel de résolution T

Dans cette section on veut déterminer la distribution $\mathcal{D}_{n_{visites}}$ sur le nombre de visites par patient, l'horizon temporel de résolution T et les jours de disponibilité des patients sur cette plage.

On considère que les patients ne peuvent pas être visités plus d'une fois par journée de disponibilité. Ainsi, on couvre un cas très général. En effet dans de nombreux cas, le patient a besoin d'être visité une seule fois par journée de visite, comme par exemple dans le contexte de l'aide à la personne âgée où celle-ci est visitée quotidiennement pour un suivi régulier.

Alayacare résout des instances sur un horizon temporel qui est de l'ordre de la semaine dans le cas général. Nous considérons donc les visites peuvent s'étaler jusqu'à 7 jours dans le temps.

En ce qui concerne les jours de disponibilité des patients, on les génère aléatoirement entre le premier jour et le 7e jour.

Génération des visites préprogrammées

Les visites préprogrammées sont déterminées selon les étapes suivantes :

— Faire tourner l'algorithme de l'$ALNS$ une première fois sur l'ensemble $\mathcal{D}'$ sans visite préprogrammée.
— Pour chaque instance I de D', on récupère la liste des visites V_I de la solution finale.
— Pour chaque instance I de D', on crée les instances I_p qui correspondent à I à laquelle on fixe à $p\%$ le nombre de visites de V_I en visites préprogrammées.

On choisit de prendre p dans les valeurs $\{0, 25, 50, 75\}$.

4.2 Configuration des paramètres pour la génération d'instances

Pour la génération de l'ensemble D nous avons fait varier les paramètres d'instances suivants :

Nombre de patients et d'aides-soignants :

- Plage de valeurs du nombre de patients : $\mathcal{N}_P = \{30, 40, 50, 60, 70, 80, 90, 100\}$
- Plage de valeurs du nombre d'aides-soignants : $\mathcal{N}_A = \{3, 6, 9, 12, 15\}$
- Ensemble $\mathcal{N}_{P,A} = \mathcal{N}_P \times \mathcal{N}_A$

Génération des localisations :

- Pour chaque couple (n_P, n_A), les localisations sont générées selon les modes : *Random*, *Clusters*$= 3$, *Clusters*$= 5$

Expertises :

- Pour chaque couple (n_P, n_A), les expertises des patients sont attribuées selon les 3 scénarios : S_1^E, S_2^E et S_3^E correspondant à un nombre maximal E_P de 1,2 et 3 expertises par patient respectivement.

Une fois que les instances sont générées, on a fait tourner l'algorithme *ALNS* sur chacune des instances sur 5000 itérations. Puis on crée les visites préprogrammées à partir des solutions :

Visites préprogrammées :

- Pour chaque couple (n_P, n_A), les visites préprogrammées sont générées à partir d'un pourcentage $p\%$ des visites de la meilleure solution pour chaque instance, avec $p \in \{0, 25, 50, 75\}$.

576 instances sont générer selon cette configuration. L'ensemble d'entraînement est de 400 instances et l'ensemble de test est de 176 instances.

4.3 Configuration des algorithmes de learning

Nous avons déterminé une configuration optimale des hyperparamètres de *ALNS-PPO* et *ALNS-SAC* par recherche en grille. La recherche en grille a été effectuée en faisant varier les valeurs des hyperparamètres sur des plages de valeur centrées sur les valeurs par défaut fournies avec les implémentations des algorithmes. Les configurations ainsi obtenues sont données dans les tableaux 4.6 et 4.7.

Tableau 4.6 Configuration de *ALNS-PPO*

Valeurs des hyperparamètres de *ALNS-PPO*	
Nom	Valeur
learning rate de l'acteur	1e-3
learning rate du critique	1e-3
gamma	0.99
taille d'un *batch*	256
nombre total de pas	1e6
nombre de mises à jour	1000
couches cachées de l'acteur	[64, 64, 64]
couches cachées du critique	[64, 64, 64]
lambda	0.95
poids sur l'entropie	1e-2
poids sur la perte du critique	5e-1

Tableau 4.7 Configuration de *ALNS-SAC*

Valeurs des hyperparamètres de *ALNS-SAC*	
Nom	Valeur
learning rate de l'acteur	1e-3
learning rate du critique	1e-3
gamma	0.99
taille d'un *batch*	256
nombre total de pas	1e6
couches cachées de l'acteur	[64, 64, 64]
couches cachées du critique	[64, 64, 64]
tau	5e-3
période du réseau	1e1
période du réseau cible	5e2

4.3.1 Résultats Expérimentaux

Production des résultats Pour comparer les modèles à *ALNS*, nous avons fait tourner les trois algorithmes sur l'ensemble D' selon les étapes suivantes :

- Pour chaque instance de D' et pour chaque scénario $S1$ et $S2$ on lance 10 *runs* de 1000 itérations pour chacun des algorithmes avec la même solution initiale. Les trois algorithmes sont lancés sur le même nombre d'itérations.
- On compare *ALNS* à *ALNS-PPO* et *ALNS* à *ALNS-SAC* sur la moyenne de ces 10 *runs*.
- Les résultats sont enregistrés pour la comparaison des performances.

Plateforme de calcul Les expériences ont été menées sur une machine équipée d'un CPU Intel(R) Core(TM) i5-8300H 2.30GHz, de 24 Go de RAM. Le système d'exploitation utilisé est Ubuntu 24.04 avec le noyau Linux 6.8.0. Les langages utilisés sont C++ pour la partie ALNS et Python pour l'apprentissage profond des modèles. La librarie Boost a été utilisée pour relier C++ à Python. La librarie Torch sous Python 2.1.2 a été utilisée pour l'apprentissage des modèles.

Temps de calcul Les temps de calcul pour l'apprentissage de *ALNS-PPO* et *ALNS-SAC* varient de 5h pour les plus petites instances ($n_P \leq 40$, $n_A \leq 9$) à 8h pour les plus grandes instances ($n_P \geq 80$, $n_A \leq 12$). Le

La comparaison des résultats se fait en trois parties décrites ci-dessous.

Performance On évalue la performance de *ALNS-PPO* et *ALNS-SAC* relativement à *ALNS* en étudiant les valeurs pour chaque composante de la fonction coût pour chacun deux scénarios $S1$ et $S2$. On rappelle que pour le scénario $S1$ la hiérarchie entre les composantes du coût est : Visites rejetées, Fidélite, Distance tandis que que pour le scénario $S2$ la hiérarchie est : Visites rejetées, Distance, Fidélité. Pour l'étude de la comparaison des performances, on utilise la métrique *Average GAP value* dont la formule est donnée par l'équation (4.6).

$$\begin{aligned} &\text{Avg-Gap-V}(x, y) = \frac{1}{N} \sum_{i=1}^{N} \frac{V(x_i) - V(y_i)}{V(y_i) + 1_{\{V(y_i)=0\}}} \\ &\text{avec } x, y \in \mathbb{R}^N \end{aligned} \tag{4.6}$$

Les valeurs affichées sont les moyennes de cette métrique sur 10 *runs*. Une valeur négative (respectivement positive) indique que le modèle a une meilleure (respectivement moins bonne) performance que *ALNS*.

Convergence et Complexité Pour comparer la vitesse de convergence des algorithmes vers la meilleure solution, nous avons utilisé la notion d'intégrale sous la courbe. Plus précisement, nous avons introduit la métrique $I_f(x)$ qui correspond à la valeur de l'intégrale discrète sous la courbe d'une fonction $f : \mathbb{R}^N \to \mathbb{R}$ appliquée à un vecteur $x \in \mathbb{R}^N$ à laquelle est soustrait $f(x_N)$. La formule de cette métrique est donnée par l'équation (4.7).

$$\begin{aligned} I_f(x) &= \sum_{t=1}^{N} \left(f(x_t) - f(x_N)\right) \\ &\text{avec } f : \mathbb{R}^N \to \mathbb{R} \text{ et } x \in \mathbb{R}^N \end{aligned} \tag{4.7}$$

La valeur $I_f(x)$ représente la vitesse de convergence de la fonction f et plus sa valeur est petite, plus la fonction f converge rapidement. Nous avons appliqué cette métrique à la fonction du coût de la meilleure solution en fonction de l'itération pour les Visites rejetées et on notera sa valeur par $I_{\text{Visites}}(x^*)$, où Visites(x^*) représente le coût en Visites rejetées du vecteur x^* des meilleures solutions sur les 1000 itérations.

De manière générale, étant donné deux algorithmes A et B, pour que l'utilisation de la

métrique $I_f(x)$ soit pertinente pour comparer A et B par rapport à la vitesse de convergence de f en fonction de x, il faut que $I_f(x)$ satisfasse la condition suivante : $f(x_0)$ doit avoir la même valeur pour chacun des algorithmes et la fonction f doit être monotone. Sans cela on ne peut pas affirmer quel algorithme converge plus rapidement. Dans notre cas la seule fonction de coût qui vérifie cette condition est le coût Visites(x^*). En effet, la solution initiale x_0 est la même pour chacun des algorithmes (et Visites(x^*) est déterministe). De plus Visites(x^*) est monotone décroissante le long des itérations car son degré de hiérarchie est le plus fort. A l'inverse, les coûts sur la Fidélité et sur la Distance des meilleures solutions ne sont pas des fonctions monotones car leur degré de hiérarchie n'est pas le plus fort et qu'on peut accepter des solutions de moins bonne qualité. C'est pourquoi nous n'avons pas appliqué la métrique $I_f(x)$ à ces coûts. Notons par ailleurs, que le coût sur les Visites rejetées est la fonction qu'on cherche à minimiser en priorité.

Nous avons ensuite appliqué la métrique *Average GAP value* à $I_{\text{Visites}}(x^*)$ pour comparer les vitesses de convergence de *ALNS-PPO* et *ALNS-SAC* relativement à *ALNS* pour chacun des scénarios $S1$ et $S2$. On a en outre mesuré ΔT, le temps moyen d'une itération par algorithme afin de comparer les vitesses de calcul des algorithmes. On a ensuite appliqué la métrique *Average GAP value* à ΔT.

Les tableaux 4.8, 4.9, 4.10 et 4.11 fournissent une comparaison des performances de *ALNS-PPO* et *ALNS-SAC* relativement à *ALNS*. La métrique utilisée est la fonction *Average GAP value* appliquée à la valeur de chaque sous-coût de la meilleure solution finale entre le modèle (*ALNS-PPO* ou *ALNS-SAC*) et *ALNS*. Pour chaque algorithme, les colonnes de gauche à droites correspondent aux Visites rejetées, à la Fidélité et à la Distance. Les valeurs des coûts sur la Distance ont été divisés par 10^3. De manière générale, les données des tableaux montrent que *ALNS* est meilleur que *ALNS-SAC* et *ALNS-PPO* sur l'ensemble D'. En outre, elles montrent aussi que *ALNS-SAC* a toujours de meilleures performances que *ALNS-PPO*.

Le tableau 4.8 fournit une comparaison des performances en fonction du nombres de patients n_P et d'aides-soignants n_A. Les instances ont été regroupées selon les trois groupes suivants : Petit, Moyen et Large contenant respectivement des instances considérées comme petites, moyennes et larges. Les comparaisons montrent qu'*ALNS* trouve des solutions de meilleure qualité pour chacune des instances et chacun des scénarios $S1$ et $S2$. En outre, *ALNS-SAC* trouve toujours des meilleures solutions que *ALNS-PPO*. De plus, la taille des instances n'a pas d'impact significatif sur les résultats. Pour les instances où *ALNS-SAC* et *ALNS* ont une performance égale sur les Visites Rejetées, *ALNS* surpasse *ALNS-SAC* sur les valeurs de Fidélité. Pour le scénario $S1$, la meilleure performance relative des modèles est celle de *ALNS-SAC* pour l'instance (50,12) pour laquelle les performances relatives sont les plus proches avec

0%, 1.7% et 5.4% pour les Visites Rejetées, Fidélité et Distance. Pour le scénario $S2$, c'est aussi SAC qui est relativement le plus proche de $ALNS$ pour l'instance (50,6) pour laquelle les valeurs sont 0%, 17.2% et -2.0%.

TABLEAU 4.8 Comparaison des performances sur le coût de la meilleure solution finale selon le nombre de patients et d'aides-soignants

		Visites, Fidélité, Distance					
		Scénario $S1$			Scénario $S2$		
		ALNS	*ALNS-PPO*	*ALNS-SAC*	*ALNS*	*ALNS-PPO*	*ALNS-SAC*
	(n_P, n_A)	$\times 10^{-3}$	Average GAP value (%)		$\times 10^{-3}$	Average GAP value (%)	
Petit	(30, 3)	56.6, 20.6, 14.1	2.8, **-5.8**, 8.6	0.0, 51.9, 3.5	62.1, 19.3, 13.2	3.6, 0.8, **-4.2**	3.1, 16.7, **-2.7**
	(30, 6)	37.9, 22.4, 22.4	3.8, **-1.7**, 1.0	1.9, 28.4, 2.0	41.6, 48.9, 32.8	7.7, 3.9, 8.4	1.6, 15.5, 5.1
	(30, 9)	28.2, 31.0, 29.3	5.0, 11.5, **-6.5**	2.5, 7.3, 0.3	35.9, 40.4, 30.0	1.5, **-3.8**, 3.8	0.0, 21.7, 3.9
	(40, 3)	62.2, 14.7, 13.7	4.2, **-5.6**, 16.3	1.4, **-8.4**, **-0.3**	69.2, 17.0, 13.1	3.5, 5.2, 5.2	2.8, 9.5, 1.5
Moyen	(30, 12)	25.4, 42.5, 34.0	8.3, 6.0, **-0.1**	4.2, 21.2, 4.9	27.4, 52.8, 31.7	4.4, 14.4, **-1.1**	1.9, 8.6, 3.5
	(40, 6)	44.8, 31.2, 25.1	4.0, 8.0, 4.2	0.0, 33.3, 4.0	46.9, 42.0, 23.7	5.2, **-6.0**, 8.9	2.0, 10.1, 5.7
	(40, 9)	41.2, 41.0, 40.2	5.0, 11.1, **-1.0**	2.4, 3.8, 0.9	41.6, 54.1, 32.1	6.8, **-1.8**, **-1.0**	2.2, 15.0, 3.5
	(40, 12)	35.8, 47.8, 34.5	4.5, 6.7, **-0.5**	2.3, 9.4, **-0.9**	37.2, 48.9, 32.8	3.8, 3.3, 13.7	1.1, 37.0, 2.8
	(50, 3)	59.2, 14.4, 13.5	1.4, 16.2, 2.4	0.0, 23.4, 2.3	50.0, 34.2, 29.5	4.9, 2.8, 7.5	0.4, 10.5, 1.4
	(50, 6)	57.8, 38.2, 29.5	5.9, 8.6, **-1.2**	2.0, 10.1, 5.7	52.5, 38.7, 24.7	2.4, 0.7, 2.5	0.0, 17.2, **-2.0**
	(50, 9)	56.9, 54.1, 40.2	6.4, 17.6, **-1.2**	2.1, 14.8, 1.0	57.1, 53.6, 39.4	1.3, **-0.9**, **-3.5**	1.0, 1.7, 1.1
Large	(50, 12)	53.4, 52.8, 31.7	2.1, 5.9, **-3.2**	0.0, 1.7, 5.4	54.2, 51.5, 30.5	5.1, **-1.9**, 0.8	2.0, 26.6, 1.7
	(60, 12)	72.9, 31.0, 25.5	5.3, **-9.9**, 0.6	1.9, 33.8, 1.1	70.8, 34.2, 26.4	6.0, **-4.5**, 8.1	1.8, **-11.0**, 3.2
	(70, 12)	74.9, 50.4, 33.6	6.2, 23.8, 0.2	0.6, 3.6, 4.0	77.0, 55.4, 26.2	3.3, 17.3, 2.1	0.0, 37.0, 2.8
	(80, 15)	82.9, 70.3, 49.6	4.8, **-7.3**, **-1.6**	4.5, 11.6, **-0.3**	88.8, 68.2, 33.3	2.6, 12.7, 5.8	1.3, 16.8, **-0.4**
	(90, 15)	95.7, 69.0, 49.3	5.4, **-2.1**, 9.9	1.6, 8.9, 3.1	94.1, 73.5, 51.4	1.4, 8.9, **-4.1**	1.0, 1.7, 1.1
	(100, 15)	121.0, 82.7, 48.8	3.2, 13.2, 1.7	1.8, 25.6, 1.8	118.5, 79.0, 50.8	2.1, 3.2, **-7.8**	3.4, 10.5, 1.4

Le tableau 4.9 fournit une comparaison des performances en fonction du mode de génération des localisations des patients et aides-soigants. Les localisations sont générées aléatoirement pour le cas *Random* et selon k clusters dans le cas $Cluster-k$. *ALNS* trouve des solutions de meilleure qualité pour chacun des modes et des scérarios $S1$ et $S2$. En outre, *ALNS-SAC* est meilleur que *ALNS-PPO* pour chacun des modes. Par ailleurs, *ALNS-SAC* et *ALNS-PPO* se rapprochent de *ALNS* en performance pour les modes par cluster. A l'inverse, leur pire performance est pour le mode *Random*. On peut interpréter ces résultats comme le fait que *ALNS* est plus robuste face à l'augmentation de la complexité de l'espace de recherche. En effet, celui-ci devient plus complexe quand la densité de localisation s'homogénéise, ce qui est le cas lorsqu'on passe d'un espace où les localisations sont réparties en clusters - un aide-soignant va préférablement visiter un même cluster pour gagner du temps - à un espace où les localisations sont aléatoires.

TABLEAU 4.9 Comparaison des performances selon le type de génération des localisations

	Visites, Fidélité, Distance					
	Scénario $S1$			Scénario $S2$		
	ALNS	*ALNS-PPO*	*ALNS-SAC*	*ALNS*	*ALNS-PPO*	*ALNS-SAC*
Localisation	$\times 10^{-3}$	Average GAP value (%)		$\times 10^{-3}$	Average GAP value (%)	
Random	42.5, 38.7, 29.5	6.8, 5.5, 2.0	2.3, 18.1, 0.9	44.8, 35.2, 31.6	4.3, **-0.3**, 2.5	1.7, 18.2, 0.7
$Cluster-3$	47.0, 35.8, 26.0	2.0, **-0.6**, 1.2	0.0, 12.7, 3.7	46.6, 32.0, 27.3	3.4, 1.4, 0.2	0.0, 15.1, 2.6
$Cluster-5$	58.8, 28.2, 23.3	1.7, 10.1,**-2.9**	0.0, 13.4, 2.6	55.6, 31.6, 25.2	1.2, 11.4, 5.6	0.4, 8.9, 4.5

Le tableau 4.10 fournit une comparaison des performances en fonction du nombre maximal d'expertises des patients E_P. *ALNS* trouve des solutions de meilleure qualité pour chacune des valeurs de E_P et pour chacun scénario $S1$ et $S2$. En outre, c'est *ALNS-SAC* qui se situe le plus proche de *ALNS*. *ALNS-PPO* généralise moins bien que *ALNS* lorsque E_P diminue. On peut interpréter ce résultat comme étant dû au fait que *ALNS-PPO* arrive moins bien que *ALNS* à chercher dans un espace plus dense en solutions, ce qui est le cas quand E_P diminue, c'est-à-dire lorsque les possibilités d'affectations entre un aide-soignant et un patient sont plus nombreuses. Pour $E_P = 1, 3$ du scénario $S1$ et pour $E_P = 2$ du scénario $S2$, *ALNS-SAC* a des performances égales en moyenne sur les Visites rejetées mais est surpassé par *ALNS* sur la Fidélité (24.6%, 14.3% et 20.9% respectivement).

TABLEAU 4.10 Comparaison des performances selon le type de génération des localisations

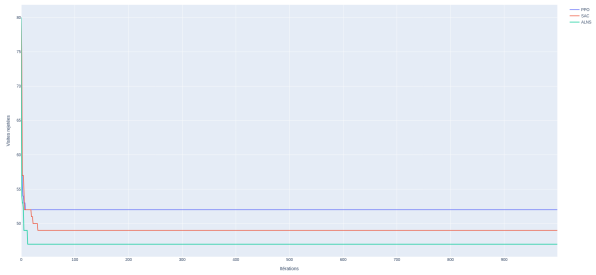
	Visites, Fidélité, Distance					
	Scénario $S1$			Scénario $S2$		
	ALNS	*ALNS-PPO*	*ALNS-SAC*	*ALNS*	*ALNS-PPO*	*ALNS-SAC*
E_P	$\times 10^{-3}$	Average GAP value (%)		$\times 10^{-3}$	Average GAP value (%)	
1	52.8, 24.6, 24.6	5.8, 7.5, **-2.5**	0.0, 24.6, 3.8	49.8, 26.4, 24.0	5.3, 7.3, **-0.4**	0.2, 21.7, 1.0
2	45.1, 26.0, 26.0	4.2, 8.9, 1.9	2.1, 8.4, 1.5	53.4, 28.8, 25.1	4.4, 8.4, 8.1	0.0, 20.9, 1.4
3	50.4, 28.2, 28.2	3.8, **-3.9**, 1.3	0.0, 14.3, 1.8	52.2, 29.1, 27.6	4.4, 6.2, 1.4	2.9, 13.2, 1.0

Le tableau 4.11 fournit une comparaison des performances en fonction du pourcentage $p\%$ de visites préprogrammées. *ALNS* trouve des solutions de meilleure qualité pour chacune des valeurs de p et pour chacun des scénarios $S1$ et $S2$. Comme pour le cas des performances en fonction de la valeur de E_P, le valeurs pour *ALNS-PPO* sont moins bonnes quand la valeur de p diminue. On peut interpréter ce résultat de manière similaire : il semble que *ALNS-PPO* est moins performant quand l'espace des solutions se densifie, ce qui est le cas quand la valeur de p diminue car le nombre de visites fixées diminue alors. La moins bonne performance de *ALNS-SAC* se situe aussi pour $p = 0$ où la valeur sur les Visites rejetées est de 2.2%.

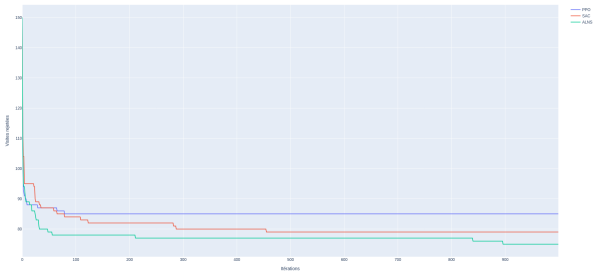
TABLEAU 4.11 Comparaison des performances selon le nombre pourcentage $p\%$ de visites préprogrammées

	Visites, Fidélité, Distance					
	Scénario $S1$			Scénario $S2$		
	ALNS	*ALNS-PPO*	*ALNS-SAC*	*ALNS*	*ALNS-PPO*	*ALNS-SAC*
$p\%$	$\times 10^{-3}$	Average GAP value (%)		$\times 10^{-3}$	Average GAP value (%)	
0%	48.6, 32.3, 25.4	6.7, 3.5, 0.3	2.2, 13.4, 2.4	48.6, 32.7, 26.1	5.0, 3.2, 1.1	1.6, 38.6, 2.4
25%	49.3, 33.5, 26.2	6.3, 8.6, 3.8	0.0, 26.9, **-4.4**	49.1, 33.9, 26.3	4.0, **-0.8**, 1.6	0.0, 7.9, 4.2
50%	49.9, 34.2, 26.4	3.8, 4.9, 0.0	1.9, 15.9, 2.2	49.5, 35.6, 25.8	3.4, 0.4, 1.7	1.0, **-1.9**, 1.0
75%	50.0, 16.8, 27.0	1.7, **-4.4**, 0.0	0.0, 3.2, 0.5	50.4, 34.0, 26.0	2.7, 3.2, 0.5	2.8, 16.5, 1.4

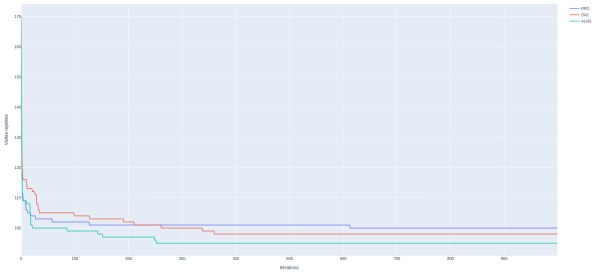
La figure 4.11 donne l'évolution du coût de la meilleure solution sur les Visites rejetées en fonction des itérations de *PPO* (bleu), *ALNS-SAC* (rouge) et *ALNS* (vert) pour 4 instances. Ces instances ces labellisées par $(n_P, n_A, E_P, \text{Localisations}, p\%, \text{Scénario})$ dont les valeurs correspondent aux paramètres de l'instance. Les figures 4.11a, 4.11b et 4.11c montrent que pour les instances associées *ALNS* est meilleur en performance que *ALNS-SAC* et *ALNS-PPO*. Pour ces instances, les coûts finaux (*ALNS-PPO*, *ALNS-SAC*, *ALNS*) sont respectivement de (52, 49, 47), (85, 79, 75) et (100, 98, 95). A l'inverse, la figure 4.11d montre que pour l'instance $(100, 15, 2, Cluster - 3, 25\%, S2)$, *ALNS-SAC* et *PPO* trouvent une meilleure solution que *ALNS*. Les coûts finaux (*ALNS-PPO*, *ALNS-SAC*, *ALNS*) sont de (122, 120, 126).



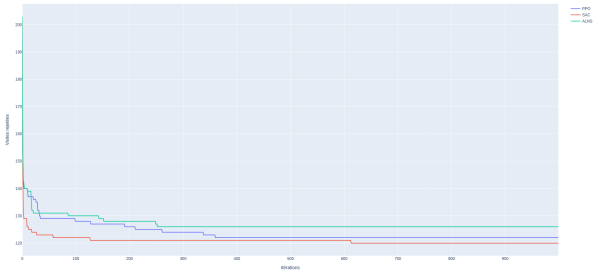
(a) $(30, 3, 2, Random, 0\%, S1)$



(b) $(70, 12, 1, Cluster - 5, 75\%, S1)$



(c) $(80, 15, 3, Random, 50\%, S2)$



(d) $(100, 15, 2, Cluster - 3, 25\%, S2)$

FIGURE 4.11 Evolution du coût sur les Visites rejetées pour différentes instances $(n_P, n_A, E_P, \text{Localisations}, p\%, \text{Scénario})$

Les tableaux 4.12, 4.14, 4.13 et 4.15 fournissent une comparaison des vitesses de convergence vers la meilleure solution finale ainsi qu'une comparaison des temps de calcul moyens de *ALNS-PPO* et *ALNS-SAC* relativement à *ALNS* pour les scénarios $S1$ et $S2$. Pour la vitesse de convergence, la métrique utilisée est Avg-GAP-I et pour le temps de calcul moyen d'une itération la métrique utilisée est Avg-GAP-ΔT. De manière générale, les données des tableaux montrent que *ALNS-PPO* converge plus rapidement vers sa valeur finale que *ALNS-SAC* et *ALNS* sur l'ensemble D'. En outre, elles montrent que *ALNS* est l'algorithme qui calcule le plus rapidement, suivi de *ALNS-PPO* puis de *ALNS-SAC*.

Le tableau 4.12 fournit une comparaison des vitesses de convergence et du temps de calcul en fonction du nombre de patients n_P et d'aides-soignants n_A. D'une part, pour chacun des scénarios $S1$ et $S2$, *ALNS-PPO* converge plus rapidement que *ALNS* tandis que *ALNS-SAC* est le plus lent à converger. Sa meilleure performance relative à *ALNS* est de 8.8%, pour le scénario $S1$ pour l'instance (40,9). Sa moins bonne performance est pour l'instance (30,3) pour laquelle il est 50% plus lent à converger que *ALNS*. En outre, la taille de l'instance ne semble pas avoir d'incidence particulière sur le temps de convergence. D'autre part, on peut

voir que *ALNS* est plus rapide sur toutes les instances et que *ALNS-PPO* est légèrement plus rapide que *ALNS-SAC* (1e-5 près) indépendamment de la taille de l'instance. Enfin on voit que plus les instances sont grandes, plus *ALNS-PPO* et *ALNS-SAC* sont lents en temps de calcul par rapport à *ALNS*. Cela peut s'interpréter par le fait que *ALNS-PPO* et *ALNS-SAC* demandent plus de ressources de calcul que *ALNS*, et bien que cela soit indépendant de la taille de l'instance, l'impact se fait plus ressentir quand les instances demandent plus de capacité de calcul.

TABLEAU 4.12 Comparaison des vitesses de convergence et des temps de selon le nombre de patients et d'aides-soignants

		$I_{\text{Visites}}(x^*)$, ΔT ($\times 10^3$)			
		Scénario $S1$		Scénario $S2$	
		ALNS-PPO	*ALNS-SAC*	*ALNS-PPO*	*ALNS-SAC*
	(n_P, n_A)	Average GAP value (%)			
Petit	(30, 3)	5.45, 1.1	55.87, 1.2	7.2, 2.1	31.4, 1.2
	(30, 6)	0.59, 1.2	34.21, 1.2	**-6.6**, 1.2	22.3, 1.3
	(30, 9)	0.04, 1.4	21.07, 1.4	**-0.8**, 1.6	35.4, 1.6
	(40, 3)	**-5.64**, 1.5	39.49, 1.7	**-10.5**, 1.6	24.2, 1.4
Moyen	(30, 12)	5.54, 1.3	14.63, 1.4	**-0.1**, 1.0	39.0, 1.9
	(40, 6)	**-6.37**, 1.6	24.81, 1.8	**-0.0**, 0.9	29.7, 1.8
	(40, 9)	**-9.55**, 1.7	8.8, 2.0	4.0, 1.5	31.6, 2.0
	(40, 12)	**-4.35**, 2.0	29.53, 2.2	**-5.9**, 1.7	43.2, 1.9
	(50, 3)	4.22, 1.7	31.53, 2.0	2.4, 1.4	22.9, 1.6
	(50, 6)	**-7.79**, 2.1	22.15, 2.2	**-2.5**, 1.8	39.5, 1.7
	(50, 9)	**-8.78**, 2.5	35.92, 2.6	**-4.9**, 2.1	29.2, 1.9
Grand	(50, 12)	**-2.0**, 0.9	28.6, 2.0	**-7.5**, 1.1	27.2, 1.8
	(60, 12)	**-2.2**, 1.9	27.9, 2.0	**-0.1**, 1.8	43.1, 2.5
	(70, 12)	**-1.7**, 1.5	25.2, 0.9	**-10.7**, 1.4	14.0, 1.5
	(80, 15)	**-8.4**, 1.5	32.9, 2.0	**-5.5**, 1.0	41.1, 1.8
	(90, 15)	**-6.8**, 1.0	14.8, 2.4	**-5.4**, 1.5	40.8, 1.3
	(100, 15)	**-5.7**, 2.2	45.9, 2.2	**-7.7**, 2.0	30.4, 1.2

Le tableau 4.13 fournit une comparaison des vitesses de convergence et du temps de calcul en fonction du nombre d'expertises maximal E_P par patient. On observe la même tendance que pour le tableau 4.12. Cependant, l'écart entre *ALNS-SAC* et *ALNS* augmente en défaveur de *ALNS-SAC* quand E_P augmente. L'écart entre *ALNS-PPO* et *ALNS* augmente en faveur de *ALNS-PPO* quand E_P augmente. Pour les temps de calcul, *ALNS-SAC* et *ALNS-PPO* sont sensiblement pareils et plus lents que *ALNS*.

TABLEAU 4.13 Comparaison des vitesses de convergence et des temps de calcul selon le nombre d'expertises maximal des patients E_P

	$I_{\text{Visites}}(x^*)$, ΔT ($\times 10^3$)			
	Scénario $S1$		Scénario $S2$	
	ALNS-PPO	*ALNS-SAC*	*ALNS-PPO*	*ALNS-SAC*
E_P	Average GAP value (%)			
1	**-1.78**, 1.7	25.12, 1.9	**-1.2**, 1.7	27.0, 1.8
2	**-2.18**, 1.6	24.46, 1.7	**-2.6**, 1.6	27.2, 1.7
3	**-4.26**, 1.8	32.54, 1.9	**-2.8**, 1.6	33.0, 1.7

Le tableau 4.14 fournit une comparaison des vitesses de convergence et du temps de calcul en fonction du mode de génération des localisations. On observe la même tendance que pour le tableau 4.12. Pour le mode de génération *Random*, *ALNS-PPO* est relativement plus rapide à converger comparé à *ALNS-SAC* et *ALNS*. *SAC* converge plus lentement que *ALNS*, jusqu'à 42%. Pour les temps de calcul, *ALNS-SAC* et *ALNS-PPO* sont sensiblement pareils et plus lents que *ALNS*.

TABLEAU 4.14 Comparaison des vitesses de convergence et des temps de calcul selon le type de génération des localisations

	$I_{\text{Visites}}(x^*)$, ΔT ($\times 10^3$)			
	Scénario $S1$		Scénario $S2$	
	ALNS-PPO	*ALNS-SAC*	*ALNS-PPO*	*ALNS-SAC*
Localisations	Average GAP value (%)			
Random	**-5.19**, 1.6	35.77, 1.7	**-4.6**, 1.8	42.4, 1.7
$Cluster-3$	**-2.73**, 1.7	19.01, 1.8	**-1.7**, 1.7	24.8, 1.9
$Cluster-5$	**-0.29**, 1.8	27.33, 2.0	**-0.9**, 1.7	24.7, 1.9

Le tableau 4.15 fournit une comparaison des vitesses de convergence et du temps de calcul en fonction du pourcentage $p\%$ de visites préprogrammées 4.12. Pour chacun des scénarios $S1$ et $S2$, *ALNS* converge toujours plus rapidement que *ALNS-SAC*, jusqu'à 42.86% pour $p\% = 50\%$. Mais *ALNS* est plus lent à converger que *ALNS-PPO* qui converge jusqu'à 4.87% plus rapidement. Les temps de calcul sont sensiblement les mêmes entre *ALNS-SAC* et *ALNS-PPO* et sont tous les deux relativement plus lents en temps de calcul que *ALNS* (de 2e-4 près).

TABLEAU 4.15 Comparaison des vitesses de convergence et des temps de calcul selon le pourcentage $p\%$ de visites préprogrammées

	$I_{\text{Visites}}(x^*)$, ΔT ($\times 10^3$)			
	Scénario $S1$		Scénario $S2$	
	ALNS-PPO	*ALNS-SAC*	*ALNS-PPO*	*ALNS-SAC*
$p\%$	Average GAP value (%)			
0%	**-4.87**, 1.7	15.76, 1.8	**-2.9**, 1.7	14.5, 1.8
25%	**-3.54**, 1.7	21.78, 1.9	**-0.8**, 1.7	24.4, 1.9
50%	**-3.47**, 1.7	42.86, 1.9	**-3.4**, 1.7	35.2, 1.8
75%	**0.93**, 1.7	29.08, 1.8	**-3.8**, 1.7	34.0, 1.8

4.4 Analyse des résultats

En terme de qualité de solution, les résultats des comparaisons montrent que *ALNS* est meilleur que les modèles *ALNS-PPO* et *ALNS-SAC* sur l'ensemble D'. Cela suggère que pour la nature de ces données, *ALNS* a une meilleure stratégie de recherche et sait plus généraliser lorsqu'on fait varier le nombre de patients et d'aides-soignants, le nombre maximal d'expertises des patients, le mode de génération de données et le pourcentage de visites préprogrammées laissées fixées.

En ce qui concerne la vitesse de convergence, *ALNS-PPO* dépasse *ALNS-SAC* et *ALNS* sur l'ensemble des données utilisé. Cette vitesse de convergence est à mitiger car *ALNS-PPO* souffre de moins bonnes performances. Cela indique notamment qu'il tombe rapidement dans des minima locaux. En outre, *ALNS-SAC* est plus lent à converger que *ALNS* ce qui le rend moins attractif car il est aussi moins performant que *ALNS*. Quant au temps de calcul, *ALNS* est également plus rapide que les modèles *ALNS-PPO* et *ALNS-SAC*. La différence est par ailleurs constante sur l'ensemble des données. *ALNS* est donc préférable si le contexte incite à chercher des solutions rapidement dans le temps. Par ailleurs, le modèle *ALNS-PPO* est légèrement plus rapide que *ALNS-SAC*. On peut interpréter cela par le fait que *ALNS-SAC* fait intervenir 2 réseaux de neurones supplémentaires par rapport à *ALNS-PPO*, ce qui alourdit les calculs.

Enfin, on peut noter le fait que *ALNS-SAC* surpasse *ALNS-PPO* sur l'ensemble D'. Cela peut être dû à la nature plus stable de *ALNS-SAC* par rapport à *ALNS-PPO* quant à la mise à jour de ses paramètres. Ce résultat laisse présumer que même si *ALNS-PPO* est un algorithme populaire et généralement robuste, dans certains contexte, d'autres algorithmes comme *ALNS-SAC* peuvent offrir de meilleures performances et avoir plus de potentiel.

CHAPITRE 5 CONCLUSION

Dans cette étude nous avons intégré du *Reinforcement Learning* (*RL*) à la méthode de recherche locale *ALNS* appliquée à un problème de planification et d'ordonnancement des soins à domicile. La partie adaptative de sélection d'opérateurs de *ALNS* a été remplacée par un modèle de *RL*. En ce qui concerne le choix de modèles utilisés, nous avons choisi un modèle *on-policy* et un modèle *off-policy*. Les algorithmes correspondant sont *ALNS-PPO* et *ALNS-SAC* respectivement. Nous avons en outre généré un ensemble de données en se basant sur un ensemble de données réelles fournit par Alayacare. La génération a été faite en faisant varier comme paramètres d'instance le nombre de patients, le nombre d'aides-soignants, le nombre d'expertises maximal des patients, le type de génération des localisations et les pourcentages de visites préprogrammées laissées fixées. La phase de tests a consisté en la comparaison des performances en qualité de solution et en temps des algorithmes *ALNS-PPO* et *ALNS-SAC* à *ALNS*.

Les résultats montrent que *ALNS* trouve des solutions de meilleure qualité par rapport à *ALNS-PPO* et *ALNS-SAC* sur l'ensemble des données étudiées. *ALNS* a su mieux généraliser aux variations des paramètres d'instance tout en étant plus rapide en temps de calcul. *ALNS-PPO* a été le plus rapide à converger mais a montré des performances inférieures et une moins bonne capacité à sortir des minima locaux. *ALNS-SAC* a été moins performant que *ALNS* mais a su montrer néanmoins une stabilité en performance par rapport à la variation des paramètres d'instances. En termes de temps de calcul, *ALNS-SAC* a été légèrement plus lent que *ALNS-PPO*.

Cette étude est exploratoire et présente plusieurs limites.

Un premier aspect limitant est l'architecture utilisée pour la conception du système de *RL*. Premièrement, la représentation de l'état d'observation peut être améliorée afin de mieux capturer l'environnement. Le choix des caractéristiques d'observation est crucial pour aider l'agent dans sa compréhension de l'environnement. Par ailleurs, la représentation a due être limitée en taille pour réduire la complexité en temps de calcul. Ceci réduit la qualité des observations et ajoute du biais à l'agent. Deuxièmement, la définition de la fonction de récompense utilisée ne répond pas forcément au mieux à l'objectif de chercher une meilleur solution.

Enfin, il est difficile de faire ressortir toutes les nuances relatives aux performances des algorithmes. Bien que les métriques utilisées sont pertinentes, elles sont spécifiques et peuvent ne pas capturer tous les aspects de performance désirés.

Une piste de solution d'amélioration porte sur l'architecture de *RL* utilisée. On pourrait explorer d'autres types de modèles comme les *Graph Convolutional Network* ou des *pointer network* pour lesquels l'observation serait un graphe représentant les patients et les aides-soignants. Une autre architecture suggérée serait celle du *Language Model*. Pour ce type de modèle, le but serait de "traduire" une observation en un *token* qui serait un couple d'opérateurs de modification de la solution. En outre, il serait intéressant d'explorer des représentations d'état plus spécifiques au problème. D'autres fonctions de récompenses pourraient aussi être investiguées pour *ALNS-PPO* et *ALNS-SAC*.

RÉFÉRENCES

[1] R. S. Sutton et A. G. Barto, *Reinforcement Learning : An Introduction*, 2e éd. The MIT Press, 2018.

[2] Statistique Canada. (2022) Estimations de la population au 1er juillet, par âge et genre.

[3] Santé Canada. (2024) Ententes en matière de santé : Priorités partagées en matière de santé.

[4] Institut canadien d'information sur la santé (ICIS). (2024) Perspectives de la population de personnes âgées au canada : du jamais vu.

[5] I. Bello, H. Pham, Q. Le, M. Norouzi et S. Bengio, "Neural combinatorial optimization with reinforcement learning," 11 2016.

[6] N. Mazyavkina, S. Sviridov, S. Ivanov et E. Burnaev, "Reinforcement learning for combinatorial optimization : A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.

[7] Y. Bengio, A. Lodi et A. Prouvost, "Machine learning for combinatorial optimization : A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, p. 405–421, 2021.

[8] Y. Peng, B. Choi et J. Xu, "Graph learning for combinatorial optimization : A survey of state-of-the-art," *Data Science and Engineering*, vol. 6, no. 2, p. 119–141, 2021.

[9] H. He, H. Daume III et J. M. Eisner, "Learning to search in branch and bound algorithms," dans *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence et K. Weinberger, édit., vol. 27. Curran Associates, Inc., 2014.

[10] G. Zarpellon, J. Jo, A. Lodi et Y. Bengio, "Parameterizing branch-and-bound search trees to learn branching policies," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, p. 3931–3939, 05 2021.

[11] Y. Tang, S. Agrawal et Y. Faenza, "Reinforcement learning for integer programming : Learning to cut," dans *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III et A. Singh, édit., vol. 119. PMLR, 13–18 Jul 2020, p. 9367–9376.

[12] E. Khalil, H. Dai, Y. Zhang, B. Dilkina et L. Song, "Learning combinatorial optimization algorithms over graphs," dans *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan et R. Garnett, édit., vol. 30. Curran Associates, Inc., 2017.

[13] M. Gasse, D. Chételat, N. Ferroni, L. Charlin et A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," dans *Neural Information Processing Systems*, 2019.

[14] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura et D. L. Dill, "Learning a SAT solver from single-bit supervision," dans *International Conference on Learning Representations*, 2019.

[15] S. Gu et T. Hao, "A pointer network based deep learning algorithm for 0–1 knapsack problem," p. 473–477, 2018.

[16] A. Barro, "Pointer networks with q-learning for op combinatorial optimization," 2024.

[17] O. Vinyals, M. Fortunato et N. Jaitly, "Pointer networks," dans *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama et R. Garnett, édit., vol. 28. Curran Associates, Inc., 2015.

[18] V. Mnih, A. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver et K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 02 2016.

[19] ravichandra addanki, V. Nair et M. Alizadeh, "Neural large neighborhood search," dans *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020.

[20] D. Pisinger et S. Ropke, "Large neighborhood search," 2010.

[21] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems." *International conference on principles and practice of constraint programming, Springer*, p. 417–431, 1998.

[22] A. Ali, K. Akhnoukh, B. Kaltenhäuser et K. Bogenberger, *Neural Network Based Large Neighborhood Search Algorithm for Ride Hailing Services*, 08 2019, p. 584–595.

[23] D. Aksen, O. Kaya, F. Sibel Salman et Özge Tüncel, "An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem," *European Journal of Operational Research*, vol. 239, n°. 2, p. 413–426, 2014.

[24] C. Chen, E. Demir et Y. Huang, "An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and delivery robots," *European Journal of Operational Research*, vol. 294, n°. 3, p. 1164–1180, 2021.

[25] E. Demir, T. Bektaş et G. Laporte, "An adaptive large neighborhood search heuristic for the pollution-routing problem," *European Journal of Operational Research*, vol. 223, n°. 2, p. 346–359, 2012.

[26] C. Friedrich et R. Elbert, "Adaptive large neighborhood search for vehicle routing problems with transshipment facilities arising in city logistics," *Computers Operations Research*, vol. 137, p. 105491, 2022.

[27] P. Grangier, M. Gendreau, F. Lehuédé et L.-M. Rousseau, "An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization," *European Journal of Operational Research*, vol. 254, nº. 1, p. 80–91, 2016.

[28] A. N. Gullhav, J.-F. Cordeau, L. M. Hvattum et B. Nygreen, "Adaptive large neighborhood search heuristics for multi-tier service deployment problems in clouds," *European Journal of Operational Research*, vol. 259, nº. 3, p. 829–846, 2017.

[29] Y. Li, H. Chen et C. Prins, "Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests," *European Journal of Operational Research*, vol. 252, nº. 1, p. 27–38, 2016.

[30] S. Ropke et D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation science*, vol. 40, nº. 4, p. 455–472, 2006.

[31] J. Kallestad, R. Hasibi, A. Hemmati et K. Sörensen, "A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems," *European Journal of Operational Research*, vol. 309, nº. 1, p. 446–468, 2023.

[32] S. S. Langfeldt, M. H. Langholm et T. H. Spangelo, "Neural network assisted large neighborhood search for personnel rostering," Mémoire de maîtrise, Master's thesis in Industrial Economics and Technology Management, June 2022.

[33] R. Reijnen, Y. Zhang, H. C. Lau et Z. Bukhsh, "Online control of adaptive large neighborhood search using deep reinforcement learning," 2024.

[34] S.-N. Johnn, V.-A. Darvariu, J. Handl et J. Kalcsics, "Graph reinforcement learning for operator selection in the alns metaheuristic," 2023.

[35] S. Levine, C. Finn, T. Darrell et P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, nº. 39, p. 1–40, 2016.

[36] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *Science*, vol. 362, p. 1140–1144, 2018.

[37] C. Hayes, R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. Zintgraf, R. Dazeley, F. Heintz, E. Howley, A. Irissappane, P. Mannion, A. Nowe, G. Ramos, M. Restelli, P. Vamplew et D. Roijers, "A practical guide to multi-objective reinforcement learning and planning," *Autonomous Agents and Multi-Agent Systems*, vol. 36, 04 2022.

[38] F. Grenouilleau, "Online control of adaptive large neighborhood search using deep reinforcement learning," 2020.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford et O. Klimov, “Proximal policy optimization algorithms,” 2017.

[40] T. Haarnoja, A. Zhou, P. Abbeel et S. Levine, “Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *ArXiv*, vol. abs/1801.01290, 2018.

[41] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta et J. G. Araújo, “Cleanrl : High-quality single-file implementations of deep reinforcement learning algorithms,” *Journal of Machine Learning Research*, vol. 23, nº. 274, p. 1–18, 2022.

[42] J. Achiam, “Spinning up in deep reinforcement learning,” 2018.

ANNEXE A ALGORITHME D'INITIALISATION DES POIDS DES OPÉRATEURS (ALNS)

Algorithm 6 InitialiserPoids

Require: Ensemble des opérateurs de destruction D et de réparation R (input)
Ensure: Poids W

function InitialiserPoids(D, R)
    $W \leftarrow \{\}$
    for all opérateur de destruction $d \in D$ **do**
        $w_d \leftarrow \frac{1}{|D|}$
        Ajouter w_d à W
    end for
    for all opérateur de réparation $r \in R$ **do**
        $w_r \leftarrow \frac{1}{|R|}$
        Ajouter w_r à W
    end for
    return W
end function

ANNEXE B ALGORITHME DE MISE À JOUR DES POIDS DES OPÉRATEURS (ALNS)

Algorithm 7 MettreAJourPoids

Require: Opérateurs de destruction d et de réparation r, score D et *gamma* (input)

function MettreAJourPoids(W, d, r, S, γ)
    for all opérateur de destruction $d' \in D$ **do**
        $w_{d'} \leftarrow$ poids W de correspondant à d'

$$w_{d'} \leftarrow \frac{w_{d'} \cdot S + 1_{\{d'=d\}} \cdot \gamma}{S + \gamma}$$

    end for
    for all opérateur de réparation $r' \in R$ **do**
        $w_{r'} \leftarrow$ poids de W correspondant à r'

$$w_{r'} \leftarrow \frac{w_{r'} \cdot S + 1_{\{r'=r\}} \cdot \gamma}{S + \gamma}$$

    end for
end function

ANNEXE C ALGORITHME DU CALCUL DE GAMMA (ALNS)

Algorithm 8 CalculerGamma

Require: Solutions courante x, candidate x', meilleure x^*, booléen $is\_accepted$ (input)
Ensure: γ (output)

function CalculerGamma(x, x', x^*, $is\_accepted$)
    $\gamma \leftarrow 0$
    if $f(x) \leq f(x')$ **and** $is\_accepted$ **then**
        $\gamma \leftarrow 1$
    else if $f(x') < f(x)$ **and** $f(x^*) \leq f(x')$ **then**
        $\gamma \leftarrow 3$
    else if $f(x') < f(x^*)$ **then**
        $\gamma \leftarrow 5$
    end if
    return γ
end function

ANNEXE D ALGORITHME DU RECUIT SIMULÉ

Algorithm 9 AccepterSolution

Require: solution courante x, solution candidate x', température T (input)
Ensure: booléen *is_accepted*

function AccepterSolution(x, x', T)
    is_accepted $\leftarrow$ **false**
    $\Delta E \leftarrow f(x') - f(x)$
    if $\Delta E < 0$ **then**
        is_accepted $\leftarrow$ **true**
    else
        $P \leftarrow \exp\left(\frac{-\Delta E}{T}\right)$
        if TirerNombreAléatoire() $< P$ **then**
            is_accepted $\leftarrow$ **true**
        end if
    end if
    MettreAJourTemperature(T)
    return *is_accepted*
end function

ANNEXE E ALGORITHME DE MISE À JOUR DE LA TEMPÉRATURE DU RECUIT SIMULÉ

Algorithm 10 Mettre à jour la température

Require: température T (input)

```
1: function MettreAJourTemperature(T)
2:     T ← α · T
3: end function
```

ANNEXE F ALGORITHME DE CALCUL DES STATISTIQUES

Algorithm 11 Calculer les statistiques d'un ensemble de trajectoires

Require: ensemble de trajectoires $\mathcal{D}$ (input)
Ensure: ensemble de moyennes $\mathbf{M}$, ensemble d'écarts-types $\mathbf{\Sigma}$ (output)

function CalculerStats($\mathcal{D}$)
    Initialiser $\mathbf{M} \leftarrow \{\}$
    Initialiser $\mathbf{\Sigma} \leftarrow \{\}$
    for all composante s^i d'une observation s **do**
        if faire une normalisation min - max
            Calculer le min min^i de s^i sur $\mathcal{D}$
            Calculer le max max^i de s^i sur $\mathcal{D}$
            Ranger min^i dans $\mathbf{M}$
            Ranger $max^i - min^i$ dans $\mathbf{\Sigma}$
        else if faire une normalisation standard
            Calculer la moyenne μ^i de s^i sur $\mathcal{D}$
            Calculer l'écart type σ^i de s^i sur $\mathcal{D}$
            Ranger μ^i dans $\mathbf{M}$
            Ranger σ^i dans $\mathbf{\Sigma}$
    end for
    return $\mathbf{M}$, $\mathbf{\Sigma}$
end function

ANNEXE G ALGORITHME D'UNE COLLECTE DE TRAJECTOIRE

Algorithm 12 Collecter une trajectoire

Require: instance u, ensemble de moyennes $\mathbf{M}$, ensemble d'écarts-types $\mathbf{\Sigma}$ (input)
Ensure: mémoire D (output)

function CollecterTrajectoire($u, \pi_\theta, \psi, \mathbf{M}, \mathbf{\Sigma}$)
    Initialiser la mémoire $D = \{\}$
    Initialiser le pas $P = \{\}$
    for $t = 0, 1, \ldots, T-1$ **do**
        $\mathcal{P} \leftarrow$ CollecterUnPas($\mathcal{P}, \pi_\theta, \psi, \mathbf{M}, \mathbf{\Sigma}$)
        Ranger le pas $\mathcal{P}$ dans D
    end for
    return D
end function

ANNEXE H ALGORITHME DE COLLECTE D'UN PAS DE TRAJECTOIRE

Algorithm 13 Collecter un pas de trajectoire

Require: pas $\mathcal{P}$, ensemble de moyennes $\mathbf{M}$, ensemble d'écarts-types $\mathbf{\Sigma}$ (input)
Ensure: pas $\mathcal{P}$ (output)

function CollecterUnPas($\mathcal{P}, \pi_\theta, \psi, \mathbf{M}, \mathbf{\Sigma}$)
    if $\mathcal{P}$ est vide **then**
        Faire un tirage de l'état initial selon la distribution initial $s \sim \rho_0(u)$
    else
        Récupérer s' de $\mathcal{P}$
        $s \leftarrow s'$
    end if
    $s \leftarrow$ NormaliserObservation($s, \mathbf{M}, \mathbf{\Sigma}$)
    Echantillonner l'action $a \leftarrow \pi_\theta(s)$
    Transiter vers l'état suivant $s' \leftarrow P(s, a)$
    Récupérer la récompense $r \leftarrow R(s, a, s')$
    $\mathcal{P} \leftarrow (s, a, r, s')$
    return $\mathcal{P}$
end function

ANNEXE I ALGORITHME DE NORMALISATION D'UNE OBSERVATION

Algorithm 14 Normaliser une observation

Require: observation s_t, ensemble de moyennes $\mathbf{M}$, ensemble d'écarts-types $\mathbf{\Sigma}$ (input)
Ensure: obervation normalisée $\widehat{s_t}$ (output)
1: **function** NormaliserObservation($s_t, \mathbf{M}, \mathbf{\Sigma}$)
2: $\widehat{s_t} \leftarrow \frac{s_t - \mathbf{M}}{\mathbf{\Sigma}}$
3: **return** $\widehat{s_t}$
4: **end function**

ANNEXE J ALGORITHME DU LAMBDA RETURN

Algorithm 15 Calculer le λ-*return*

Require: trajectoire τ, critique ψ_ϕ (input)
Ensure: λ-*return* R^λ (output)

function CalculerLambdaReturn(τ, ψ_ϕ)
    Initialiser $R^\lambda \leftarrow \{\}$
    for pas $t = 1, 2, \ldots, T$ **do**
        for $n = 1, 2, \ldots, T$ **do**
            Calculer le n-*steps returns* $R_t^n \leftarrow \sum_{k=0}^{n-1} \gamma^k \, r_{t+k} + \gamma \, \psi_\phi(s_{t+n})$
        end for
        Calculer le λ-*return* $R_t^\lambda \leftarrow (1-\lambda) \cdot \sum_{n=1}^{T} \lambda^{n-1} R_t^n$
        Ranger R_t^λ dans R^λ
    end for
    return R^λ
end function

ANNEXE K ALGORITHME DE LA GENERALIZED ADVANTAGE ESTIMATION

Algorithm 16 Calculer le *Generalized Advantage Estimation*

Require: trajectoire τ, λ-*return* R^{λ}, critique ψ_{ϕ} (input)
Ensure: $\hat{A}_t$ (output)

function CalculerGAELambda(τ, ψ_{ϕ})
    Initialiser $\hat{A} \leftarrow \{\}$
    for pas $t = 1, 2, \ldots, T$ **do**
        $\hat{A}_t \leftarrow R_t^{\lambda} - \psi_{\phi}(s_t)$
        Ranger $\hat{A}_t$ dans $\hat{A}$
    end for
    return $\hat{A}$
end function

ANNEXE L ALGORITHME DE LA LOSS SUR L'ACTEUR (ALNS-PPO)

Algorithm 17 Calculer la *loss* sur l'acteur (*ALNS-PPO*)

Require: acteur π_θ, $\pi_{\theta_{old}}$, mini-batch $\boldsymbol{b}_\tau$, avantages $\hat{A}$ (input)
Ensure: *loss* $L^\pi(\theta)$ (output)

function CalculerActorLoss($\pi_\theta, \pi_{\theta_{old}}, \boldsymbol{b}_\tau, \hat{A}$)
  for all pas t de $\boldsymbol{b}_\tau$ **do**
    Calculer le rapport de probabilité d'obtenir a_t sachant s_t :

$$r_t(\theta) \leftarrow \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}$$

  end for
  Calculer la CLIP loss :

$$L^{\text{CLIP}}(\theta) \leftarrow \mathbb{E}_{t\sim\boldsymbol{b}_\tau}\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

  Calculer la *loss* sur l'entropie

$$H(\theta) \leftarrow \mathbb{E}_{t\sim\boldsymbol{b}_\tau}\left[-\sum_{a\in\mathcal{A}} \pi_\theta(a|s_t)\log\left(\pi_\theta(a|s_t)\right)\right]$$

  Calculer la *loss* pondérée de l'acteur :

$$L^\pi(\theta) \leftarrow -L^{\text{CLIP}}(\theta) - c_H \cdot H(\theta)$$

  return $L^\pi(\theta)$
end function

ANNEXE M ALGORITHME DE LA LOSS SUR LE CRITIQUE (ALNS-PPO)

Algorithm 18 Calculer la *loss* sur le critique

Require: critique ψ_ϕ, mini-batch $\boldsymbol{b}_\tau$, λ-*return* R^λ (input)
Ensure: *loss* $L^{\mathrm{V}}(\phi)$ (output)
1: **function** CalculerCriticLoss($\psi_\phi, \boldsymbol{b}_\tau, R^\lambda$)
2: Calculer la *loss* sur le critique :

$$L^{\mathrm{V}}_{\mathrm{VANILLA}}(\phi) \leftarrow \mathbb{E}_{t\sim \boldsymbol{b}_\tau}\left[(R_t(\lambda) - \psi_\phi(s_t))^2\right]$$

$$L^{\mathrm{V}}_{\mathrm{CLIP}}(\phi) \leftarrow \mathbb{E}_{t\sim \boldsymbol{b}_\tau}\left[\mathrm{clip}\left(\psi_\phi(s_t), \psi_{\phi_{old}}(s_t) - \epsilon, \psi_{\phi_{old}}(s_t) + \epsilon\right) - R^\lambda_t)^2\right]$$

$$L^{\mathrm{V}}(\phi) \leftarrow max\left(L^{\mathrm{V}}_{\mathrm{VANILLA}}(\phi),\ L^{\mathrm{V}}_{\mathrm{CLIP}}(\phi)\right)$$

3: **return** $L^{\mathrm{V}}(\phi)$
4: **end function**

ANNEXE N ALGORITHME DE LA LOSS SUR L'ACTEUR (ALNS-SAC)

Algorithm 19 Calculer la *loss* de l'acteur (*ALNS-SAC*)

Require: batch $\boldsymbol{b}_{\mathcal{D}}$ (input)
Ensure: *loss* $L^{\pi}(\theta)$ (output)
1: **function** CalculerActorLoss($\boldsymbol{b}_{\mathcal{D}}$)
2:
$$L^{\pi}(\theta) \leftarrow -\mathbb{E}_{t\sim \boldsymbol{b}_{\mathcal{D}}}\left[\mathbb{E}_{a_t\sim\pi_\theta}\left[\min_{i=1,2} Q_{\phi_i}(s_t, a_t) - \alpha \log\left(\pi_\theta(a_t \mid s_t)\right)\right]\right]$$
3: **return** $L^{\pi}(\theta)$
4: **end function**

ANNEXE O ALGORITHME DE LA LOSS SUR LE CRITIQUE (ALNS-SAC)

Algorithm 20 Calculer la *Q-loss*

Require: batch $\boldsymbol{b}_{\mathcal{D}}$ (input)
Ensure: *Q-loss* L^{Q} (output)

function CalculerQLoss($Q_{\phi_{1,2}}, \boldsymbol{b}_{\mathcal{D}}$)
  for chaque pas t dans $\boldsymbol{b}_{\mathcal{D}}$ **do**
    Calculer le *Q-target* :

$$y_t \leftarrow r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi_\theta} \left[\min_{i=1,2} Q_{\phi_i'}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\theta(a_{t+1} \mid s_{t+1}) \right]$$

  end for

$$L^{Q_{\phi_i}} \leftarrow \mathbb{E}_{t \sim \boldsymbol{b}_{\mathcal{D}}} \left(Q_{\phi_i}(s_t, a_t) - y_t \right)^2 \quad \text{pour } i \in \{1, 2\}$$

$$L^{Q}(\phi_1, \phi_2) \leftarrow L^{Q_{\phi_1}} + L^{Q_{\phi_2}}$$

  return $L^{Q}(\phi_1, \phi_2)$
end function

ANNEXE P ALGORITHME DE LA LOSS SUR LA TEMPÉRATURE

Algorithm 21 Calculer la *loss* sur la température α

Require: batch $\boldsymbol{b}_{\mathcal{D}}$ (input)
Ensure: *loss* $L(\alpha)$ (output)
1: **function** CalculerAlphaLoss($\boldsymbol{b}_{\mathcal{D}}$)
2:
$$L(\alpha) \leftarrow \mathbb{E}_{t\sim\boldsymbol{b}_{\mathcal{D}}}\left[-\alpha \cdot \mathbb{E}_{a_t\sim\pi_\theta}\left[\log\left(\pi_\theta(a_t|s_t)\right) + \bar{H}\right]\right]$$
3: **return** $L(\alpha)$
4: **end function**