SORBONNE
UNIVERSITÉ

Travail encadré de recherche

Master Mathématiques et applications

2024-2025

Optimisation robuste de tournées de vehicules : Le consistent routing

Leïna Montreuil
Sorbonne université
leina.montreuil@etu.sorbonne-universite.fr

Anne-Claire Haury
LPSM - OR Team

Table des matières

Introduction et présentation du problème

La croissance rapide des besoins en logistique dans des domaines variés comme la livraison de colis, le transport de marchandises ou la gestion de tournées de véhicules de service, a donné lieu à de nombreux défis d'optimisation. Parmi eux, le *Problème de Tournée de Véhicules* (ou *Vehicle Routing Problem*, VRP) occupe une place centrale. Formulé pour la première fois par Dantzig and Ramser (1959) dans un contexte logistique concret : l'optimisation des tournées de livraison de carburant. Leur objectif était de déterminer des itinéraires efficaces pour une flotte de camions-citernes, de manière à satisfaire la demande de plusieurs stations-service tout en minimisant la distance totale parcourue. Ce problème a été présenté comme une généralisation du célèbre problème du voyageur de commerce (TSP). Ce cadre a donné les bases d'un domaine de recherche majeur en optimisation combinatoire et en recherche opérationnelle, donnant naissance à de nombreuses variantes adaptées aux besoins logistiques modernes. Dans manière général, ce problème modélise une situation courante : comment organiser efficacement les itinéraires d'une flotte de véhicules pour desservir un ensemble de clients. L'objectif est de minimiser un coût global (souvent la distance parcourue ou le temps de service), tout en respectant diverses contraintes opérationnelles telles que la capacité des véhicules (CVRP) ou des fenêtres de livraison (VRPTW).
Plus précisément, le VRP consiste à déterminer, pour un nombre donné de véhicules partant d'un ou plusieurs dépôts, quels clients doivent être visités, par quel véhicule, et dans quel ordre. Chaque client doit être servi exactement une fois, et les tournées doivent respecter les contraintes du problème.
Bien que cette formulation paraisse intuitive, le VRP appartient à la classe des problèmes *NP-difficiles*. Cela signifie que le temps de résolution croît rapidement avec le nombre de clients, rendant le problème presque intractable à grande échelle par des méthodes exactes.

Le VRP se décline en de nombreuses variantes, en fonction des contraintes pratiques ou des objectifs opérationnels visés. Voici une liste de quelqes variantes :

- **Capacitated VRP (CVRP)** : chaque véhicule possède une capacité maximale, et la somme des demandes sur chaque tournée ne doit pas la dépasser
- **VRP with Time Windows (VRPTW)** : chaque client doit être visité dans un créneau horaire donné, ce qui introduit des contraintes temporelles supplémentaires sur les tournées
- **Stochastic VRP** : certains paramètres du problème (demandes, temps de trajet, disponibilité des clients) sont aléatoires ou incertains, ce qui requiert des solutions robustes ou adaptatives
- **Consistent VRP (conVRP)** : les tournées sont planifiées sur plusieurs jours, et l'objectif est de garantir une certaine régularité (ou cohérence) dans les itinéraires d'un jour à l'autre, par exemple pour améliorer la qualité de service ou la satisfaction

des clients.

L'étude porte ainsi sur cette dernière variante, centrée sur la *cohérence temporelle et structurelle des tournées*, analysée dans un contexte stochastique.

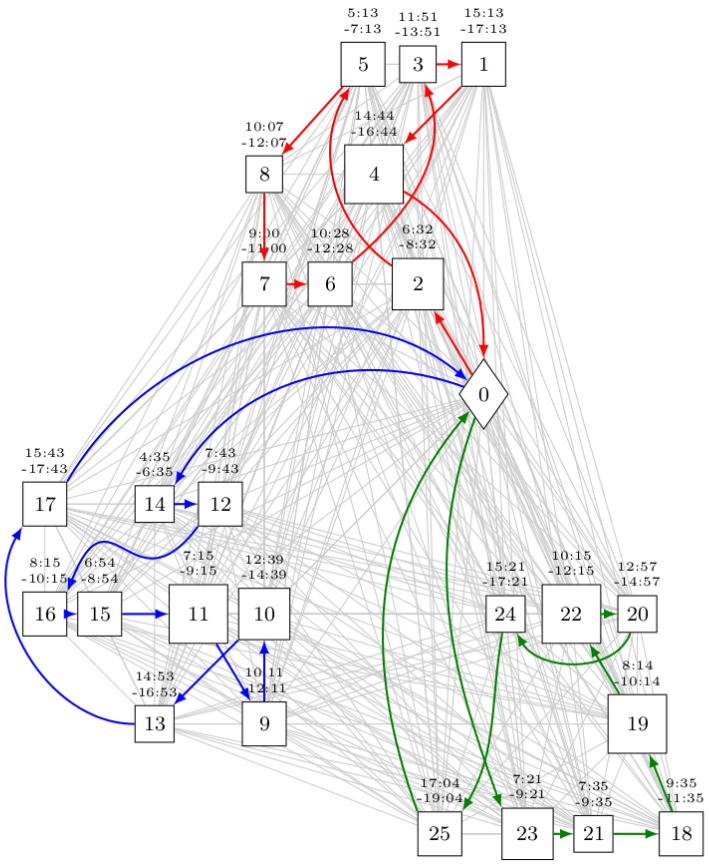


FIGURE 1 – Exemple d'instance CVRPTW (Capacitated VRP with Time Windows) avec 25 clients et un dépôt, construite à partir des 26 premiers nœuds de l'instance `rc201` du benchmark de Solomon (1987)

.

La Figure 1 illustre une instance du problème de *Capacitated Vehicle Routing Problem with Time Windows* (CVRPTW). Dans cette variante, chaque tournée doit respecter à la fois une **contrainte de capacité** des véhicules et une **fenêtre temporelle** propre à chaque client.
Les nœuds représentent les clients, dont la taille est proportionnelle à leur demande, et sont annotés avec leurs fenêtres de temps respectives. La distance euclidienne entre deux nœuds i et j sert à la fois à calculer le coût c_{ij} et le temps de trajet τ_{ij} associé à l'arête (i, j). Les arêtes représentées en bleu, vert et rouge correspondent aux trois tournées optimales trouvées, minimisant le coût total tout en satisfaisant les contraintes imposées. L'illustration est extraite de la thèse de Bono (2020).

Au-delà de la seule efficacité logistique, certaines applications exigent une forme de régularité dans les tournées de livraison. C'est le cas, par exemple, dans les services de livraison hebdomadaire (exemple des pharmacies), le transport scolaire, ou encore les tournées médicales à domicile. Ces cas d'usage ont conduit à la définition du Consistent Vehicle Routing Problem (conVRP), formulé pour la première fois par Groër et al. (2009), une extension du VRP classique intégrant des contraintes de stabilité inter-périodes.

Dans un conVRP, le problème n'est plus d'optimiser une tournée unique, mais de planifier une séquence de tournées sur plusieurs jours, de manière à minimiser les variations dans les itinéraires des véhicules en respectant les contraintes de capacité et de coût.
L'objectif est de garantir une certaine stabilité des tournées : par exemple, un client donné doit idéalement être visité par le même véhicule, à des horaires similaires, ou dans un ordre proche, d'un jour à l'autre. Cette régularité renforce la qualité de service, la satisfaction client, et facilite l'organisation opérationnelle.
En d'autres termes, le conVRP cherche un compromis entre optimisation du coût logistique et robustesse opérationnelle. Ce que l'on perd en flexibilité à court terme, on le regagne en efficacité sur le terrain à long terme : agents plus efficaces, fidélisation des clients, diminution des erreurs et des retards.

La popularité du VRP se reflète également dans la littérature scientifique. Le Vehicle Routing Problem (VRP) est un problème largement étudiés en recherche opérationnelle. Comme le montre la Figure 2, le nombre de publications consacrées au VRP a connu une croissance exponentielle ces dernières années.

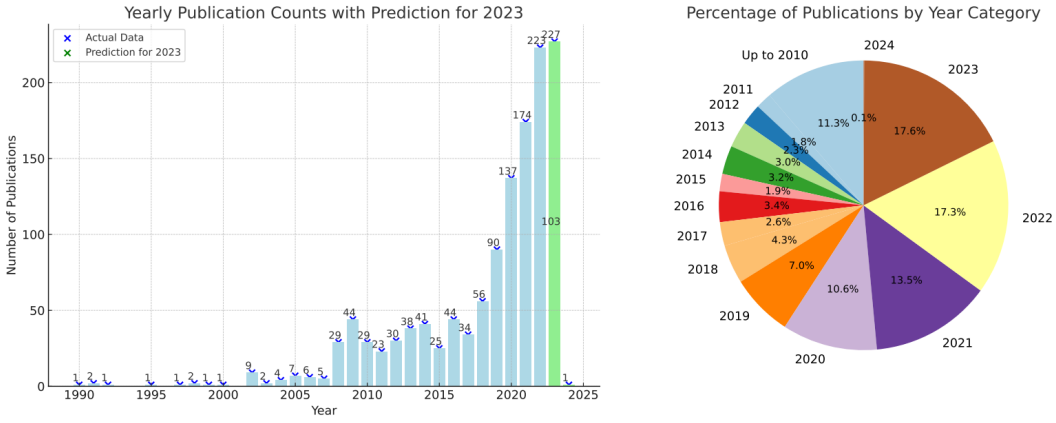


FIGURE 2 – Évolution du nombre de publications liées au VRP indexées dans Scopus avec les mots-clés « Vehicle Routing Problem » et « Learning » Shahbazian et al. (2024).

La figure de gauche présente l'évolution annuelle du nombre d'articles publiés depuis 1990, avec une prédiction pour 2023, tandis que la figure de droite répartit les publications par tranches d'années. On y observe clairement une accélération notable après 2021, qui concentre à elle seule près de 50% des publications répertoriées. Cette tendance souligne non seulement la pertinence académique du sujet, mais également l'intérêt croissant pour des approches apprenantes, en particulier dans des variantes comme le conVRP.

Formulation d'un problème de VRP Le *Vehicle Routing Problem* (VRP) classique peut être formulé sur un graphe complet $G = (V, E)$, où :

- $V = \{0, 1, \ldots, n\}$ est l'ensemble des sommets, avec 0 représentant le dépôt et les nœuds représentent les clients
- E est l'ensemble des arêtes entre les sommets qui modélisent un trajet,
- c_{ij} est le coût (ou la distance) associé à l'arête (i, j),
- d_i est la demande du client i,
- Q est la capacité maximale de chaque véhicule,
- K est l'ensemble des véhicules disponibles.

On introduit les variables de décision suivantes :

$$x_{ij}^k = \begin{cases} 1 & \text{si le véhicule } k \text{ emprunte l'arête } (i,j), \\ 0 & \text{sinon.} \end{cases}$$

L'objectif est de minimiser la distance totale parcourue par l'ensemble des véhicules :

$$\min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}^k$$

Sous les contraintes suivantes :

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \qquad \forall i \in V \setminus \{0\} \quad \text{(chaque client est visité une fois)} \tag{1}$$

$$\sum_{j \in V} x_{ij}^k = \sum_{j \in V} x_{ji}^k \qquad \forall k \in K,\ \forall i \in V \quad \text{(véhicule entre et sort du même client)} \tag{2}$$

$$\sum_{i \in V} d_i \left(\sum_{j \in V} x_{ij}^k \right) \leq Q \quad \forall k \in K \quad \text{(capacité)} \tag{3}$$

Cette formulation constitue une base pour modéliser le VRP classique. Dans la suite, nous introduirons des extensions pour capturer la régularité temporelle des tournées, propre au conVRP.

Ce mémoire s'intéresse à une version stochastique et multi-période du *Consistent VRP* (conVRP), dans laquelle certains clients sont fixes (présents chaque jour), tandis que d'autres apparaissent de manière ponctuelle, simulant un environnement incertain mais réaliste. Dans ce contexte, notre objectif est double : explorer l'efficacité d'une approche par apprentissage dans un environnement stochastique et analyser l'impact de différents paramètres du modèle sur la qualité des solutions.

Remerciements

Je souhaite tout d'abord exprimer mes sincères remerciements à mon encadrante, Anne-Claire Haury, qui a accepté sans hésitation de me superviser lorsque je lui ai proposé. Je la remercie pour sa patience, sa confiance, ses conseils avisés, la qualité de son accompagnement, ainsi que le temps qu'elle m'a consacré tout au long du semestre. Sa disponibilité constante et sa bienveillance m'ont été d'un grand soutien.

Je tiens également à remercier mon directeur d'études, Ismaël Castillo, qui m'a encouragée à postuler pour un TER et dont le soutien a été déterminant dans cette démarche. Sans lui, je n'aurais probablement pas osé me lancer dans cette aventure.

Je remercie aussi les responsables pédagogiques Laurent Boudin et Julien Grivaux, pour avoir validé mon inscription à ce TER et pour la confiance qu'ils m'ont accordée.

Par ailleurs, je souhaite remercier ma famille pour leur regard extérieur, leurs soutiens et leurs retours critiques sur mon travail, qui m'ont aidée à prendre du recul sur mes choix et ma progression. Je tiens à adresser une mention particulière à mon frère Yannis, ancien étudiant à Jussieu et aujourd'hui doctorant, dont les travaux de recherche portent sur des thématiques comme la robustesse dans l'apprentissage automatique. Ses conseils éclairés, ses remarques pertinentes, ainsi que ses questions ont grandement enrichi ma réflexion et m'ont permis d'améliorer à la fois ma compréhension du sujet et la rigueur de ma démarche. Les nombreux échanges que nous avons eus, parfois techniques, parfois stratégiques, ont représenté un appui précieux tout au long de ce TER.

Notations - Abréviation

TABLE 1 – Liste des abréviations utilisées dans le mémoire

Abréviation	**Signification**
VRP	Vehicle Routing Problem (Problème de tournée de véhicules)
ConVRP	Consistent Vehicle Routing Problem
CVRP	Capacitated Vehicle Routing Problem (VRP avec contraintes de capacité)
RL	Reinforcement Learning (Apprentissage par renforcement)
MDP	Markov Decision Process (Processus de décision de Markov)
MILP	Mixed Integer Linear Programming (Programmation linéaire en nombres entiers mixtes)
NN	Nearest Neighbor (Plus proche voisin)
VNS	Variable Neighborhood Search
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit (type de RNN)
LSTM	Long Short-Term Memory (type de RNN)
ML	Machine learning

Chapitre 1

Le Consistent VRP

Le *Consistent Vehicle Routing Problem* (conVRP ou encore le problème de tournées cohérentes), introduit par Groër et al. (2009), étend le VRP classique en intégrant des considérations de **stabilité** et de **régularité inter-journalière** dans la planification des tournées. Ce problème a été formulé dans un contexte, où certaines entreprises, comme celles du secteur de la livraison de colis ou des services réguliers, souhaitent que les mêmes conducteurs visitent les mêmes clients à des horaires similaires, afin de renforcer la satisfaction client, faciliter le travail des chauffeurs et améliorer l'efficacité du service.
Dans ce travail, nous nous intéressons à une version stochastique et multi-période du conVRP, motivée par la volonté de concevoir des tournées à la fois **robustes face aux perturbations** et **représentatives des contraintes pratiques** observées dans des domaines tels que la livraison à domicile ou les soins réguliers à la personne.

Dans notre approche, nous distinguerons deux types de clients :

- **Clients fixes** : présents chaque jour, ils forment le cœur stable du réseau de distribution. En pratique ils représentent des patients à visiter tous les jours, des pharmacies, des restaurants, des laboratoires médicaux...
- **Clients ponctuels** : présents de manière aléatoire, ils modélisent l'incertitude quotidienne.

L'objectif est alors de planifier, pour plusieurs jours successifs, des tournées qui s'adaptent aux variations tout en maintenant une certaine **stabilité structurelle** pour les clients fixes. Cela se traduit par la préservation d'**arcs fréquents ou récurrents** dans les tournées journalières, une idée que l'on peut formaliser via la notion d'*itinéraires templates*.

Dans notre approche, nous retenons une architecture basée sur un environnement **multi-jours avec clients fixes et clients ponctuels générés aléatoirement à chaque épisode**. La cohérence temporelle des tournées est encouragée via un **terme de consistance** (du point de vue des livreurs), intégré dans la fonction de récompense, mesurant la proportion d'arcs entre clients fixes qui sont réutilisés d'un jour à l'autre.
On considère plutôt une consistance du point de vue des livreurs car plusieurs études ont montré que la constance des itinéraires présente également des avantages du point de vue des livreurs. En effet, lorsqu'un même chauffeur suit régulièrement les mêmes tournées, il acquiert une meilleure connaissance des lieux desservis : il sait où se garer, reconnaît les entrées des bâtiments, anticipe les contraintes locales, et développe des habitudes de parcours efficaces. Cette familiarité améliore non seulement la fiabilité du service, mais contribue également à une réduction des erreurs et des temps d'arrêt. À long terme, cela se traduit par une diminution du coût global d'exploitation (Yao et al., 2021). Ces observations justifient l'importance de prendre en compte la régularité des tournées dans les

systèmes de planification modernes.

Cette formulation permet d'optimiser non seulement le coût logistique, mais aussi des objectifs tels que :

- la **stabilité des tournées** (moins de changements),
- la **fidélisation des clients** (même livreur, même horaire),
- la **satisfaction des clients, patients** (et meilleur suivi lorsque que c'est le même soignant, aux même horaires)
- et la **spécialisation des agents** sur des secteurs réguliers.

Formulation d'un probème de ConVRP Nous proposons ici une formulation de type partitionnement d'ensemble pour le Consistent Vehicle Routing Problem (conVRP) avec une pénalisation sur la non-consistance des arcs entre clients fixes tiré de Yao et al. (2021).

Notations

- S : ensemble des jours considérés.
- K : ensemble des véhicules.
- P_k^s : ensemble des chemins faisables pour le véhicule k au jour s.
- c_p : coût (distance) associé au chemin p.
- a_{ip} : vaut 1 si le client i est desservi dans le chemin p, 0 sinon.
- b_{ijp} : nombre de fois où l'arc (i, j) est parcouru dans le chemin p.
- $A^* \subseteq V \times V$: ensemble des arcs entre clients fixes.
- $\lambda_{pk}^s \in \{0, 1\}$: 1 si le véhicule k utilise le chemin p au jour s.
- $y_{ijk} \in \{0, 1\}$: 1 si le véhicule k emprunte l'arc $(i, j) \in A^*$ chaque jour.
- $\theta > 0$: paramètre de pondération de la consistance.

Formulation

$$\min \quad \sum_{s \in S} \sum_{k \in K} \sum_{p \in P_k^s} c_p \lambda_{pk}^s - \theta \cdot |S| \cdot \sum_{(i,j) \in A^*} \sum_{k \in K} y_{ijk} \tag{1.1}$$

Sous les contraintes :

$$\sum_{k \in K} \sum_{p \in P_k^s} a_{ip} \lambda_{pk}^s = 1, \qquad \forall i \in N^s,\ s \in S \tag{1.2}$$

$$\sum_{p \in P_k^s} b_{ijp} \lambda_{pk}^s \geq y_{ijk}, \qquad \forall (i, j) \in A^*,\ s \in S,\ k \in K \tag{1.3}$$

$$\lambda_{pk}^s \in \{0, 1\}, \qquad \forall p \in P_k^s,\ s \in S,\ k \in K \tag{1.4}$$

$$y_{ijk} \in \{0, 1\}, \qquad \forall (i, j) \in A^*,\ k \in K \tag{1.5}$$

La fonction objectif (1.1) cherche à minimiser le coût total des tournées sur tous les jours, tout en maximisant la réutilisation des arcs entre clients fixes (mesurée via les variables

y_{ijk}). La contrainte (1.2) garantit que chaque client est servi une fois par jour. La contrainte (1.3) définit l'activation des variables de consistance sur les arcs utilisés : l'arc (i,j) peut être considéré comme consistant (ie $y_{ijk} = 1$ activé) pour le véhicule k seulement si au moins un chemin sélectionné pour ce véhicule au jour s emprunte effectivement cet arc. Enfin, (1.4) et (1.5) imposent les contraintes d'intégralité sur les variables de décision.

Complexité Le *Consistent Vehicle Routing Problem* (ConVRP) appartient à la classe des problèmes NP-difficiles. En théorie de la complexité, cela signifie qu'il est au moins aussi difficile que les problèmes de la classe NP, pour lesquels une solution peut être vérifiée efficacement (en temps polynomial), mais dont on ne connaît pas nécessairement de méthode efficace pour les résoudre.
Dans le cas du ConVRP, la complexité est double : d'une part, il hérite de la difficulté intrinsèque du VRP classique, lui-même NP-difficile, en raison du grand nombre de permutations possibles des clients et des contraintes de capacité. D'autre part, il ajoute une dimension temporelle avec des exigences de régularité inter-journalière (stabilité des arcs ou des affectations clients-véhicules), ce qui accroît encore la taille et la structure de l'espace des solutions.
Il devient alors compliqué de résoudre de manière exacte des instances de taille réaliste en particulier dans un contexte stochastique. Cela justifie pleinement le recours à des approches heuristiques, à des métaheuristiques ou encore à des méthodes par apprentissage automatique, capables de produire des solutions de qualité en un temps raisonnable.

Chapitre 2

Etat de l'art

Dans cette section, nous présentons un panorama des approches classiques de résolution VRP et de ses variantes, en particulier le ConVRP

D'après mes recherches, aucune méthode d'apprentissage automatique n'a encore été proposée spécifiquement pour résoudre le conVRP dans sa forme générale. Par conséquent, l'état de l'art que nous dressons ici repose d'une part sur les méthodes existantes spécifiquement développées pour le conVRP lorsqu'elles sont disponibles, et d'autre part sur les approches utilisées pour résoudre le VRP classique, qui peuvent potentiellement être adaptées au cadre du conVRP.

Nous distinguons les quatre grandes familles de méthodes suivantes :

— les **méthodes exactes**, qui cherchent à résoudre le problème de manière optimale en explorant l'espace des solutions de façon exhaustive ou branchée (ex : programmation linéaire en nombres entiers). Elles sont précises mais limitées à des instances de petite taille en raison de leur complexité.

— les **méthodes heuristiques**, qui utilisent des règles simples ou des algorithmes gloutons pour construire rapidement des solutions valables mais non garanties optimales (ex : insertion, savings). C'est une idée simple, intuitive qui fonctionne mais qui peut être vraiment amélioré

— les **métaheuristiques**, qui emploient des stratégies d'exploration plus sophistiquées pour améliorer les solutions heuristiques en évitant les minima locaux (ex Tabu Search).

— les **méthodes par apprentissage**, qui cherchent à apprendre un comportement décisionnel, une politique ou une heuristique directement à partir de données ou par interaction avec l'environnement. Elles sont prometteuses pour généraliser à de nouvelles instances tout en réduisant les temps de calcul en phase d'inférence (ex PointerNetwork).

2.1 Méthodes exactes

Methodes Branch-price La méthode *Branch-and-Price* est une technique exacte puissante, utilisée dans plusieurs travaux récents pour résoudre des variantes complexes du Vehicle Routing Problem (VRP), notamment le *Consistent VRP* (ConVRP). Elle combine deux approches classiques :

— la **programmation linéaire en nombres entiers** (via un schéma de *Branch-and-Bound*),

— la **génération de colonnes dynamique** (*Column Generation*), utilisée pour gérer efficacement un très grand nombre de tournées faisables.

Dans le cadre du **ConVRP avec contrainte de cohérence des chemins** (*path consistency*), Yao et al. (2021) proposent une approche *Branch-and-Price* dans laquelle chaque colonne représente une tournée faisable pour un véhicule sur une journée donnée et une pénalisation est introduite pour les chemins trop différents entre jours consécutifs, modélisant ainsi la régularité inter-journalière. Sa formulation est celle proposée dans la section sur la Formulation du ConVRP1
De façon similaire, Tarantilis et al. (2012) développent une méthode exacte adaptée à un ConVRP avec clients fixes et clients ponctuels, où la consistance est incorporée via un coût dans la fonction objectif.

La modélisation repose généralement sur une reformulation en *Set Partitioning*, dans laquelle :

— le **problème maître restreint** *Master problem* choisit, parmi les tournées générées, un sous-ensemble optimal qui couvre tous les clients.

— le **sous-problème de génération de colonnes** est résolu comme un problème de plus court chemin avec contraintes de ressources.

Lorsque la relaxation continue du modèle est trop lâche, c'est-à-dire lorsque la solution du problème linéaire relâché autorise des solutions fractionnaires irréalistes. On renforce le modèle en ajoutant des **coupes valides**. Cela conduit à la méthode dite de **Branch-and-Cut-and-Price**, plus robuste mais aussi plus coûteuse à implémenter.

Remarque. *Lorsqu'on détend un problème en nombres entiers (ex. : $x \in \{0, 1\}$) en autorisant les variables à être continues (ex. : $x \in [0, 1]$), on obtient une relaxation continue. Celle-ci est dite* ***trop lâche*** *(loose) lorsque la solution optimale obtenue est très éloignée de la solution entière optimale : par exemple, si elle contient des variables fractionnaires ou autorise des tournées irréalistes. On utilise alors des coupes valides pour éliminer ces solutions tout en conservant les solutions entières.*

Limites des méthodes exactes. Malgré leur capacité à garantir l'optimalité, les méthodes exactes, telles que celles basées sur la programmation en nombres entiers ou le *Branch-and-Price*, souffrent d'une complexité computationnelle exponentielle. Leur utilisation devient rapidement inapplicable pour des instances de taille réaliste (plusieurs dizaines de clients sur plusieurs jours), en particulier dans des variantes comme le ConVRP, qui intègrent des dimensions temporelles, structurelles, ou stochastiques supplémentaires. Ces limites ont conduit au développement de méthodes plus flexibles et rapides, capables de fournir des solutions de qualité en un temps raisonnable, sans garantie d'optimalité. C'est dans cette perspective que s'inscrivent les **méthodes heuristiques**, que nous présentons à présent.

2.2 Méta-heuristiques et heurtistiques

Meta-heuristiques Certaines approches comme celle de Haugland et al. (2007) s'appuient sur des stratégies de recherche locale améliorée, comme la méthode *Multistart*. Elle consiste à générer plusieurs solutions initiales de manière aléatoire ou structurée, à les améliorer localement, puis à sélectionner la meilleure (comparaison des coûts). Bien que relativement simple, cette méthode permet d'explorer efficacement l'espace de solution lorsqu'elle est couplée à des heuristiques de construction robustes.

Une approche plus avancée et largement utilisée est la *Tabu Search*, une métaheuristique classique introduite par Glover and Laguna (1999), qui repose sur une mémoire tabou pour interdire temporairement certains mouvements récemment effectués. Cette mémoire joue un rôle central : elle empêche l'algorithme de revenir vers des solutions précédentes, évitant ainsi les cycles et facilitant l'évasion des minima locaux. Concrètement, à chaque itération, l'algorithme explore le voisinage de la solution courante, sélectionne la meilleure solution admissible, même si celle-ci est moins bonne que la précédente, puis met à jour la liste tabou qui interdit certains mouvements (par exemple, échanger deux clients).

Haugland et al. (2007) compare la méthode Tabu Search à une méthode Multistart dans le cadre du VRP avec demandes stochastiques et partitionnement en districts. Ils montrent que la Tabu Search parvient à obtenir des solutions significativement meilleures, en particulier sur les instances complexes où les demandes varient fortement entre scénarios.

Ces stratégies sont particulièrement pertinentes dans des contextes comme le *ConVRP*, où l'objectif est non seulement d'optimiser les tournées quotidiennes mais aussi de stabiliser les structures inter-journalières. La Tabu Search proposé dans un cadre conVRP peut, par exemple, intégrer dans sa mémoire tabou des arcs fréquents entre clients fixes, afin de favoriser la cohérence structurelle ou, au contraire, tester activement des variantes pour mieux équilibrer coût et régularité.

Méthodes heuristiques classiques. Les heuristiques classiques sont des méthodes simples, rapides et souvent utilisées pour générer des solutions initiales au VRP. Elles reposent sur des règles déterministes permettant de construire une tournée sans nécessairement garantir son optimalité. Parmi les plus connues, on peut citer la méthode de *plus proche voisin* (Nearest Neighbor), où le prochain client à visiter est toujours celui le plus proche, ou encore les heuristiques d'*insertion* (insertion la plus proche, insertion la moins coûteuse), qui insèrent les clients un à un à la position optimisant le coût. Par exemple, Tabu search peut s'appuyer sur des heuristiques d'insertions pour tester des combinaison, calculer les coûts et ainsi faire les mises à jour
Ces méthodes sont très efficaces en termes de temps de calcul et sont souvent utilisées comme point de départ pour des métaheuristiques plus complexes. Bien qu'elles ne garantissent pas la qualité optimale des solutions, elles restent utiles pour des instances de grande taille ou comme base pour l'exploration locale.

Heuristique de Clarke & Wright (Savings). Proposée en 1964, l'heuristique de Clarke & Wright repose sur l'idée d'économiser de la distance en fusionnant des tournées individuelles. À l'initialisation, chaque client est desservi séparément (dépôt $\rightarrow$ client $\rightarrow$ dépôt). Puis, pour chaque paire de clients i, j, on calcule une valeur s_{ij} (représente la valeur d'économie obtenue si l'on relie directement les clients i et j plutôt que passer par le dépôt) : où c_{ab} désigne la distance (coûts) entre les nœuds a et b

$$s_{ij} = c_{0i} + c_{0j} - c_{ij}$$

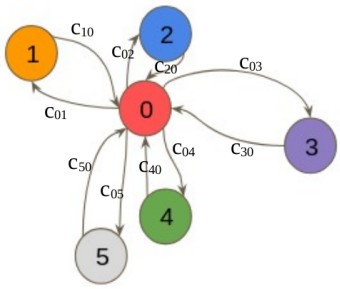


FIGURE 2.1 – Illustration du calcul des gains d'économies s_{ij} dans l'heuristique de Clarke et Wright. Le dépôt est représenté par le nœud 0, et les flèches indiquent les coûts associés aux trajets entre nœuds.

Les paires avec les économies les plus élevées sont fusionnées en priorité, tant que cela respecte les contraintes de capacité. Cette méthode est très rapide et produit des solutions de qualité raisonnable, souvent utilisées comme point de départ pour des métaheuristiques.

Méthodes de recherche locale une limitation classique des approches naïves est le risque de rester bloqué dans un minimum local. Pour pallier cela, plusieurs stratégies ont été développées afin de diversifier la recherche et d'explorer plus largement l'espace des solutions : Les méthodes de recherche locale sont des approches d'optimisation qui partent d'une solution initiale et l'améliorent itérativement en explorant son voisinage. Elles sont particulièrement utiles pour affiner des solutions ou échapper à des minima locaux dans des problèmes complexes comme le VRP. Parmi elles, la *Variable Neighborhood Search* (VNS) Mladenović and Hansen (1997) explore successivement plusieurs structures de voisinage pour favoriser la diversification, tandis que la *Large Neighborhood Search* (LNS) (Shaw (1998)) repose sur une stratégie de "destruction-reconstruction" partielle de la solution pour explorer efficacement des zones éloignées de l'espace de recherche. On note aussi que TabuSearch est à la fois une méta heuristique et une méthode de recherche locale. Ces méthodes sont souvent intégrées dans des algorithmes hybrides ou utilisés comme phases d'amélioration après une construction initiale

Remarque. *Comme le soulignent Haugland et al. (2007), un* ***prétraitement du graphe des données*** *peut s'avérer essentiel pour améliorer l'efficacité des heuristiques comme la Tabu Search. Dans leur travail sur le districting pour le VRP avec demandes stochastiques, les auteurs proposent une transformation du graphe initial, appelée adjacency graph, qui restreint les affectations possibles entre clients et véhicules. Concrètement, seuls les clients géographiquement proches sont autorisés à être desservis par un même véhicule dans une même tournée. Ce filtrage est réalisé en construisant un graphe d'adjacence basé sur la distance euclidienne, réduit l'espace de recherche et permet de guider la Tabu Search vers des solutions plus réalistes tout en évitant qu'elle ne s'enferme dans des optima locaux non pertinents.*

Limites des métaheurisitques et heuristiques Ces approches sont appréciées pour leur simplicité, leur rapidité d'exécution et leur capacité à fournir des solutions initiales raisonnables mais présentent quelques problème (Toth and Vigo, 2014; Shahbazian et al., 2024) :

— **Qualité de solution non garantie** : les heuristiques constructives (par insertion, plus proche voisin, savings, etc.) ne fournissent aucune garantie de performance.

Elles peuvent générer des solutions très éloignées de l'optimum, en particulier dans les instances avec structure irrégulière ou contraintes multiples

— **Sensibilité à l'ordre d'entrée** : de nombreuses heuristiques sont déterministes et sensibles à l'ordre initial des clients, ce qui peut entraîner des minima locaux de mauvaise qualité.

— **Manque d'exploration, d'apprentissage** : ces méthodes appliquent des règles locales, sans stratégie systématique d'exploration de l'espace des solutions. ces méthodes sont généralement conçues pour une instance à la fois. Elles ne "réapprennent" pas des problèmes précédents, ce qui limite leur usage dans des contextes dynamiques ou répétés.

2.3 Outils de résolution exacts et plateformes d'optimisation

Les problèmes de Vehicle Routing peuvent être résolus à l'aide de solveurs d'optimisation exacts, mais également via des plateformes combinatoires intégrant des heuristiques industrielles puissantes. Voici un aperçu des outils les plus utilisés dans la littérature et les applications pratiques.

Solveurs d'optimisation mathématique.

— **CPLEX (IBM ILOG)** : solveur commercial performant pour les programmes linéaires et en nombres entiers. Utilisé dans de nombreux travaux sur le VRP, avec des interfaces Python, C++, et Java.

— **Gurobi** : l'un des solveurs les plus rapides pour les problèmes MILP. Il propose une API très intuitive en Python et une licence gratuite pour les chercheurs académiques.

— **SCIP** : solveur open-source pour la programmation linéaire et non linéaire en nombres entiers. Très utilisé dans la recherche.

Outil combinatoires.

— **Google OR-Tools** : bibliothèque open-source développée par Google, spécialisée dans l'optimisation combinatoire. Elle offre des solveurs intégrés pour le VRP, TSP, CVRP, avec prise en compte de nombreuses contraintes (fenêtres de temps, capacités, multi-dépôts...). OR-Tools intègre des heuristiques (savings, insertion), des métaheuristiques (Tabu Search, Guided Local Search).

2.4 Apprentissage

Les méthodes d'apprentissage automatique appliquées au VRP ont récemment connu un développement important. Elles représentent une alternative prometteuse aux approches exactes et (méta)heuristiques classiques, notamment dans des contextes dynamiques ou à grande échelle. Shahbazian et al. (2024) proposent une revue des approches d'apprentissage intégrées au VRP, que nous reprenons ici.

Vue d'ensemble des architectures apprenantes pour le VRP

— **Sequence-to-Sequence (Seq2Seq)** : modèle encodeur-décodeur classique. L'entrée (liste des clients) est encodée en un vecteur de contexte fixe, puis décodée pas à pas pour prédire une séquence de visite. Limitée aux sorties sur un vocabulaire fixe.

— **Pointer Networks** : extension du modèle Seq2Seq introduite par Vinyals et al. (2017) pour produire des permutations d'entrée. Le décodeur génère des probabilités sur les éléments de l'entrée (clients), et les sélectionne par un mécanisme d'attention, ce qui les rend idéales pour les problèmes comme le TSP ou le VRP. Nous détaillerons plus tard cette méthode puisqu'elle constituera notre approche 3.4

— **Attention Mechanism (AM)** : technique permettant au modèle de "se concentrer" sur certaines parties de l'entrée lors de la génération de la sortie. Cela améliore la pertinence des décisions locales en tenant compte de l'information globale. Utilisé à la fois dans les Transformers et dans les architectures de routing par pointer. On détaillera davantage son rôle 3.4

— **Graph Neural Networks (GNN)** : familles de réseaux neuronaux conçues pour traiter directement des données en structure de graphe. Dans le contexte du VRP, les clients et les trajets sont représentés comme des nœuds et des arêtes, permettant au modèle de capturer efficacement la structure spatiale et relationnelle du problème. Parmi les variantes notables, on trouve les GCN (Graph Convolutional Networks) qui appliquent une convolution sur les voisins avec poids uniformes, et les GAT (Graph Attention Networks) qui utilisent des poids différenciés via un mécanisme d'attention.

Approches par renforcement (RL) Ces méthodes modélisent le VRP comme un processus de décision séquentiel, typiquement via un *Markov Decision Process* (MDP). L'objectif est d'apprendre une politique $\pi(a|s)$ 3.1 qui guide les choix d'actions (par exemple, le prochain client à visiter) afin de maximiser une fonction de récompense (minimisation du coût, stabilité, etc.).

Méthodes Policy Gradient. Les méthodes de type *Policy Gradient* constituent une sous-catégorie fondamentale de l'apprentissage par renforcement. Elles apprennent directement les paramètres θ d'une politique stochastique $\pi_\theta(a|s)$, sans passer par une estimation explicite de la fonction de valeur. L'approche REINFORCE Williams (1992) en est l'illustration la plus simple. Elle repose sur l'optimisation du rendement attendu par descente de gradient, et peut être améliorée par des techniques de réduction de variance comme l'introduction d'une baseline. Ces méthodes sont particulièrement adaptées au VRP, car elles permettent de générer une séquence d'actions (ordre de visite) tout en recevant un signal de récompense global à la fin d'un épisode (tournée).

Distinction entre Seq2Seq et Pointer Network. Les architectures *Sequence-to-Sequence (Seq2Seq)* ont été initialement développées pour des tâches de traduction automatique (Sutskever et al., 2014), dans lesquelles une séquence d'entrée est encodée, puis décodée mot à mot en une séquence de sortie de taille éventuellement différente. Elles supposent un vocabulaire de sortie fixe, ce qui limite leur usage dans les problèmes combinatoires comme le TSP ou le VRP.
Or, dans ces problèmes, la sortie n'est pas une séquence de mots, mais une permutation sur les éléments de l'entrée (ex. : ordre de visite des clients). Pour répondre à cette limite, Vinyals et al. (2017) ont introduit les *Pointer Networks*, une extension des modèles Seq2Seq dotée d'un mécanisme d'attention spécifique.
Contrairement au Seq2Seq classique, dans un Pointer Network, le décodeur ne sélectionne pas un mot dans un vocabulaire statique, mais pointe directement vers un élément de l'entrée, via un mécanisme d'attention. Cela permet de produire dynamiquement des sorties valides pour des problèmes où la taille et les éléments du vocabulaire dépendent de

l'instance.

Résumé des différences :

- *Seq2Seq* : produit une séquence d'éléments dans un ensemble fixe (si les demandes changent, Seq2seq n'est plus adapté).
- *Pointer Network* : génère une séquence d'indices pointant vers les éléments d'entrée.

On peut aussi citer d'autres méthodes qui s'appuient sur de l'apprentissage supervisé ou non supervisé pour assister ou améliorer des techniques classiques de résolution :

- **Arbres de décision** pour guider les règles de branchement dans des solveurs exacts.
- **Clustering (K-means)** pour regrouper les clients en sous-problèmes de routage.

Limite des approches par apprentissages L'apprentissage permet de s'affranchir de certaines limites des méthodes traditionnelles (temps de résolution, flexibilité, adaptabilité aux contextes stochastiques). Toutefois, ces approches demandent un entraînement long, de nombreuses données, et ne garantissent pas toujours l'optimalité. Shahbazian et al. (2024) observent que les modèles de type RL surpassent souvent les heuristiques classiques sur des instances de grande taille, mais restent encore peu compétitifs sur les petits cas.

Conclusion le choix de l'approche dépend étroitement de la nature du problème de VRP considéré et des contraintes spécifiques à modéliser (taille eds instances, environnement stochastique, mémoire...). Malgré les limites identifiées, j'ai choisi de me concentrer sur une méthode par apprentissage automatique, d'une part parce que ces approches restent peu, voire pas du tout, explorées dans le cadre spécifique du Consistent VRP, et d'autre part par intérêt personnel pour les méthodes d'apprentissage statistique.

Chapitre 3

Approche et méthode

Dans cette section, nous présentons l'approche adoptée pour résoudre le Consistent Vehicle Routing Problem dans un cadre stochastique. Je fais le choix d'utiliser une méthode d'apprentissage par renforcement profond s'appuyant sur une architecture actor-critic, dans laquelle l'acteur est implémenté sous la forme d'un Pointer Network.

3.1 Modélisation du problème : formulation en MDP

Le *Vehicle Routing Problem* (VRP) et ses variantes, comme le *Consistent VRP* étudié ici, peuvent être formulés naturellement comme un **Processus Décisionnel de Markov** (*MDP*). À chaque étape, l'agent (le véhicule) prend une décision (choisir un client ou retourner au dépôt) en fonction de l'état actuel de l'environnement (positions, demandes, charge restante, etc.).

Un MDP est défini par le triplet $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P)$, où :

- $\mathcal{S}$ est l'ensemble des **états** : dans notre cas, chaque état encode les positions des clients et du dépôt, les demandes restantes, la charge du véhicule, etc.
- $\mathcal{A}$ est l'ensemble des **actions** : sélectionner le prochain nœud à visiter (client ou dépôt).
- $\mathcal{R}$ est la **fonction de récompense** : elle mesure la qualité d'une tournée (typiquement, la somme des distances parcourues, avec ou sans pénalités de consistance).
- P est la **fonction de transition**

L'objectif est de trouver une **politique** π qui maximise l'espérance des récompenses cumulées.

Définition : Une **politique** π est une fonction qui guide un agent dans le choix des actions à effectuer en fonction de l'état actuel du système. C'est une **distribution de probabilité** sur les actions possibles, sachant l'état actuel :

$$\pi(a_t \mid s_t) = P(A_t = a_t \mid S_t = s_t)$$

où :

- s_t est l'état actuel de l'environnement (séquences de clients déjà visités, client sur lequel il est actuellement)
- a_t est une action possible à partir de cet état (prochain client à visiter).
- $\pi(a_t \mid s_t)$ représente la probabilité de choisir l'action a_t sachant que l'agent est dans l'état s_t.

3.2 Approche d'apprentissage par renforcement : architecture Actor-Critic

L'approche explorée dans ce travail est une méthode d'**apprentissage par renforcement profond** (*Deep Reinforcement Learning*) utilisant une architecture **actor-critic**. Deux réseaux de neurones sont utilisés :

— L'**actor** (politique π_θ) est un réseau de neurones paramétré par θ, entraîné à générer les tournées de livraison sous forme de séquences d'actions, en interagissant avec l'environnement. Il prend en entrée l'état s de l'environnement et génère, à chaque étape, une distribution de probabilité sur les actions possibles.

Dans notre cas, l'actor est implémenté sous la forme d'un encodeur–décodeur avec attention, inspiré des *Pointer Networks*. L'encodeur (convolutif) transforme les représentations statiques et dynamiques des nœuds en embeddings (représentation vectorielle comprise par l'encodeur) . Le décodeur (de type RNN) génère la tournée pas à pas, en produisant à chaque étape un vecteur de contexte utilisé par le mécanisme d'attention pour sélectionner le prochain nœud à visiter. L'actor s'appuie donc sur une communication explicite entre l'encodeur et le décodeur, via le module d'attention, pour construire les séquences d'actions.

— Le **critic** (valeur V_ϕ) est un réseau de neurones paramétré par ϕ, entraîné à évaluer la qualité de la tournée produite, en estimant la récompense attendue à partir de l'état courant. Il est optimisé en minimisant l'erreur quadratique entre la récompense réelle et sa propre estimation :

$$\mathcal{L}_{\text{critic}}(\phi) = \left(R - V_\phi(s)\right)^2 .$$

Une illustration est proprosé à la Figure 3.1

Paramètres du critic et de l'actor.

— Les **paramètres de l'actor**, notés θ, regroupent l'ensemble des poids et biais utilisés pour encoder les informations statiques et dynamiques de l'environnement (via des couches convolutives), le réseau récurrent du décodeur, ainsi que le mécanisme d'attention du pointeur.

— Les **paramètres du critic**, notés ϕ, définissent les poids et biais d'un réseau convolutif 3.2 ou dense, utilisé pour approximer la valeur d'un état, c'est-à-dire l'espérance des récompenses futures :

$$V^\pi(s) \approx V_\phi(s)$$

Ce signal est utilisé comme *baseline* pour réduire la variance dans la mise à jour des paramètres de l'actor. Le critic est entraîné en minimisant une erreur quadratique entre la valeur prédite $V_\phi(s)$ et la récompense observée

Ces deux ensembles de paramètres θ et ϕ sont optimisés simultanément au cours de l'entraînement, l'un guidant la sélection d'actions (actor), l'autre fournissant une estimation du retour (critic) utilisée pour améliorer cette sélection.
L'architecture utilisée est inspirée du travail Nazari et al. (2018), avec un encodeur convolutionnel, un décodeur RNN (GRU), et un mécanisme d'attention de type *pointer network*.

Remarque sur les architectures utilisées. Le modèle utilise deux types de réseaux de neurones :

— Un **GRU** (Gated Recurrent Unit) dans le décodeur de l'actor, qui permet de modéliser la dépendance temporelle entre les décisions successives. Il s'agit d'un type de RNN capable de mémoriser un historique compact des étapes précédentes tout en évitant le problème du vanishing gradient.
— Un **réseau convolutif 1D** (Conv1D) dans les encodeurs statique et dynamique, qui agit sur les nœuds (clients, dépôt) en parallèle. Ce choix permet d'extraire des représentations invariantes à l'ordre d'entrée des clients : propriété essentielle dans le VRP où les clients ne sont pas ordonnés a priori.

3.3 Principe de la méthode *Policy Gradient*

Le **policy gradient** est une famille de méthodes qui permet d'optimiser directement les paramètres θ d'une politique 3.1 $\pi_\theta(a_t \mid s_t)$, en maximisant l'espérance des récompenses cumulées.

$$\text{Soit une trajectoire : } \tau = (s_1, a_1, s_2, a_2, \ldots, s_{|\tau|})$$

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[R(\tau) \right], \quad J(\theta) = \sum_\tau \pi_\theta(\tau) R(\tau)$$

où R est la récompense obtenue en suivant la politique (sous la distribution) π_θ.

La mise à jour des paramètres s'effectue par descente de gradient, à l'aide de l'algorithme de REINFORCE Williams (1992) illustré dans l'algorithme 1

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a \mid s) \cdot R \right].$$

Afin de réduire la variance du gradient estimé, on introduit une **baseline** $b(s)$ (typiquement $V(s)$, estimée par le critic), ce qui donne la forme :

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a \mid s) \cdot (R - V(s)) \right],$$

où le terme $A(s, a) = R - V(s)$ est appelé **avantage** (*advantage function*).

Ce terme mesure à quel point une action a été meilleure ou pire que ce que le critic attendait. Cela permet de renforcer les bonnes décisions et d'affaiblir les mauvaises.

Cette méthode permet un apprentissage entièrement auto-supervisé, basé uniquement sur le signal de récompense observé, et s'adapte naturellement à des problèmes complexes comme le conVRP.

La plupart des éléments présentés dans cette section sur les méthodes de policy gradient sont inspirés du cours du M2 DAC (Lamprier, 2019).

Baseline. Dans les méthodes de policy gradient, une *baseline* est une fonction $b(s)$ qui est soustraite à la récompense observée lors du calcul du gradient, sans introduire de biais. Elle permet de réduire significativement la variance de l'estimateur de gradient, ce qui stabilise et accélère l'apprentissage.
Dans notre approche, la baseline est fournie par le **critic**, un réseau qui estime la valeur $V(s)$ attendue depuis l'état courant. Ainsi, au lieu de renforcer une action proportionnellement à la récompense totale R, on l'évalue selon son *avantage* :

$$A(s, a) = R - V(s),$$

ce qui indique si l'action a été meilleure ou pire que prévu.

Lien entre Policy Gradient et Actor-Critic L'architecture *Actor-Critic* repose sur le principe fondamental du *Policy Gradient*, selon lequel on met à jour les paramètres de la politique π_θ dans la direction qui augmente l'espérance de récompense cumulée.
Dans ce cadre, l'**acteur** (Actor) correspond à la politique $\pi_\theta(a \mid s)$, qui est responsable de la prise de décision.
De son côté, le **critique** (Critic) estime la qualité de l'état courant (ou de l'état-action) via une fonction de valeur $V(s_t)$ ou $Q(s_t, a_t)$, et permet ainsi de fournir un signal d'apprentissage à l'acteur.
C'est à ce moment que le *Policy Gradient* intervient : la mise à jour des paramètres θ de la politique est effectuée en utilisant le gradient de la log-probabilité de l'action choisie, pondéré par un estimateur de l'avantage.

Interprétation des signaux d'apprentissage

Dans cette sous-section, nous expliquons comment interpréter les sorties observées pendant l'entraînement du modèle, notamment le reward moyen, la loss moyenne, et la fonction d'avantage. Il est essentiel de bien comprendre les signaux d'apprentissage produits lors de l'entraînement, car ils permettent non seulement de diagnostiquer les éventuels dysfonctionnements du modèle, mais aussi d'ajuster efficacement les hyperparamètres. En particulier, dans notre cas, ces signaux sont cruciaux pour calibrer le coefficient de *consistance* et évaluer son impact sur le compromis entre stabilité des tournées et performance opérationnelle.

Reward R La récompense est généralement défini comme l'opposé de la distance totale de la tournée :

$$R = -\text{distance totale}$$

Un reward plus **élevé** signifie une tournée plus courte, donc meilleure.

Avantage $A(s, a)$ L'avantage mesure à quel point l'action prise est meilleure que ce que prévoyait le critic $criticEst$:

$$A(s, a) = R - V(s) = R - criticEstimation$$

- **Si** $A(s, a) > 0$: l'action a rapporté un reward meilleur que prévu. Le gradient pousse alors à auugmenter la probabilité de cette action. C'est ce qu'on appelle dans ce cas une **bonne action**
- **Si** $A(s, a) < 0$: l'action a été **moins bonne que prévu**. Le gradient pousse à **diminuer** la probabilité de cette action.
- **Si** $A(s, a) \approx 0$: l'action est conforme aux attentes du critic, donc peu ou pas de changement.

Loss (fonction objectif) La loss dans REINFORCE avec baseline est donnée par :

$$\mathcal{L} = -\mathbb{E}_{\pi_\theta}[A(s, a) \cdot \log \pi_\theta(a \mid s)]$$

Elle mesure à quel point les actions prises sont alignées avec l'avantage estimé :

- Si une action a un **avantage positif** $A(s, a) > 0$ et que sa **probabilité** $\pi_\theta(a \mid s)$ **est grande**, alors $\log \pi_\theta(a \mid s)$ est proche de 0, et la loss est peu pénalisée (bon apprentissage).

— Si une bonne action ($A(s,a)$»0) a une **faible probabilité**, alors $\log \pi_\theta(a \mid s)$ est très négatif, ce qui rend la loss **fortement positive** (mauvais apprentissage).

— Une **loss négative** signifie que le modèle a attribué de hautes probabilités à de bonnes actions, ce qui est le but de l'optimisation.

— Une **loss positive** indique que le modèle donne peu de probabilité aux bonnes actions ou beaucoup aux mauvaises.

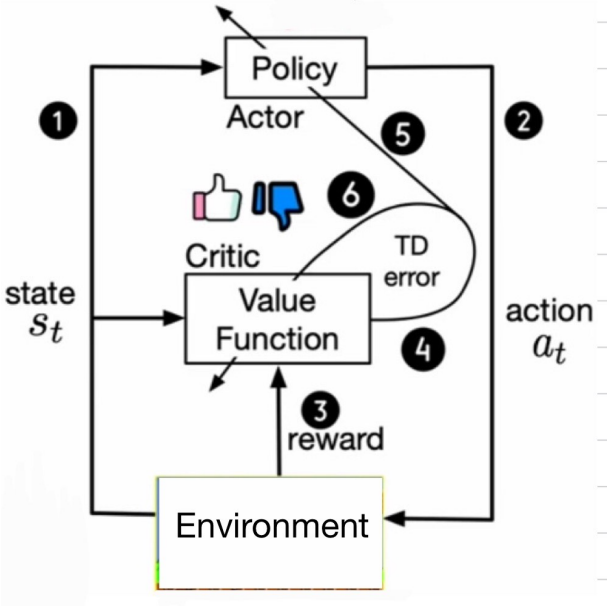


FIGURE 3.1 – Schéma de l'architecture Actor-Critic. L'Acteur propose une action, le Critique évalue la valeur de l'état courant et donne un feedback à l'actor pour qu'il s'améliore. L'illustration a été tiré de Weng (2022) et modifié.

Description des étapes :

1. **Observation de l'état** s_t : l'agent perçoit l'état actuel de l'environnement (positions, demandes, charge...).
2. **L'acteur choisit une action** a_t selon la politique $\pi_\theta(a_t|s_t)$.
3. **Interaction avec l'environnement** : l'action a_t est exécutée, ce qui génère une récompense r_t et un nouvel état s_{t+1}.
4. **Le critique évalue** $V(s_t)$ à partir de la fonction de valeur actuelle.
5. **Le TD error est calculé**. IL mesure à quel point la valeur estimée d'un état était correcte après avoir reçu une récompense et observé le prochain état. C'est l'équivalent de notre fonction avantage A(s,a)
6. **Signal d'apprentissage** :
   — Le **Critique** est mis à jour pour réduire l'erreur
   — L'**Acteur** est mis à jour par descente de gradient

3.3.1 Algorithme REINFORCE avec Critique

L'algorithme suivant correspond à la version implémentée dans ce travail, inspirée de Nazari et al. (2018) :

Algorithm 1: Algorithme REINFORCE avec Critique

Entrées : Politique π_θ, fonction de valeur V_ϕ
Initialisation : θ, ϕ

1. Pour chaque itération :

1.1 Réinitialiser les gradients $\nabla_\theta \leftarrow 0$, $\nabla_\phi \leftarrow 0$

1.2 Échantillonner N instances de problèmes

1.3 Pour chaque instance n :

— Générer une séquence d'actions Y^n selon la politique π_θ

— Calculer la récompense totale R^n

— Estimer l'avantage :

$$A^n = R^n - V(X_0^n; \phi)$$

— Accumuler les gradients :

$$\nabla_\theta \mathrel{+}= A^n \cdot \nabla_\theta \log \pi_\theta(Y^n \mid X_0^n)$$

$$\nabla_\phi \mathrel{+}= \nabla_\phi \, (A^n)^2$$

1.4 Mettre à jour θ et ϕ par montée et descente de gradient

3.4 Pointer Network

Introduits par Vinyals et al. (2017), les **Pointer Networks** sont des modèles où la sortie est une séquence d'indices pointant vers les éléments de l'entrée. Contrairement aux modèles classiques de traduction, ils permettent de gérer une sortie de longueur variable, et surtout de *pointer* vers des éléments spécifiques de l'entrée.

Soit une entrée $\mathcal{X} = \{x_1, \ldots, x_n\}$, où chaque $x_i \in \mathbb{R}^d$ représente, par exemple, les coordonnées d'un client dans un VRP. Le but est de générer une permutation $(y_1, \ldots, y_T)$ des indices correspondant aux clients à visiter.

Le modèle produit une distribution conditionnelle sur les séquences de décisions :

$$\pi_\theta(y_1, \ldots, y_T \mid x_1, \ldots, x_n) = \prod_{t=1}^{T} \pi_\theta(y_{t+1} \mid Y_t, X_t)$$

À chaque étape t, le modèle doit sélectionner le prochain nœud y_{t+1} à visiter, en se basant sur les décisions précédentes $Y_t = \{y_0, \ldots, y_t\}$ et l'état courant X_t de l'environnement (positions, demandes, charge restante, etc.). Cette sélection est modélisée par la probabilité conditionnelle :

$$\pi_\theta(y_{t+1} = i \mid Y_t, X_t) = \text{softmax}(u_t^i),$$

qui représente la probabilité, selon la politique π_θ, de choisir le nœud i (le client i) comme **prochaine action à l'étape** $t + 1$.

où le score u_t^i pour chaque entrée x_i est obtenu via un mécanisme d'attention :

$$u_t^i = v^\top \tanh(W_1 e_i + W_2 d_t)$$

avec :

— e_i : représentation (embedding) de l'entrée x_i (position, demande),

— d_t : état caché du décodeur RNN à l'étape t (résumé du contexte passé),

— v, W_1, W_2 : paramètres appris de l'attention .

Le score u_t^i mesure la pertinence de l'entrée x_i comme prochaine action, et le softmax transforme ces scores en une distribution de probabilité sur les nœuds. Ce mécanisme permet au modèle de sélectionner dynamiquement un élément de l'entrée.

Remarque. *Dans ce cadre, les entrées $x_i \in \mathbb{R}^d$, pour $i = 1, \ldots, n$, représentent les éléments fixes du problème (clients), tandis que les sorties $y_t \in \{1, \ldots, n\}$, pour $t = 1, \ldots, T$, correspondent à une séquence ordonnée d'indices d'entrée sélectionnés par le modèle. Les indices i parcourent les objets d'entrée, tandis que t indexe les étapes de décision séquentielle. La longueur de la séquence T peut être différente de n selon la structure du problème (par exemple, retours au dépôt ou recharges).*

Structure. Un modèle type Pointer Network est composé de :

— un **encodeur** (souvent convolutif ou récurrent), qui transforme les nœuds d'entrée (par ex., positions et demandes des clients) en représentations latentes e_i ;

— un **décodeur** récurrent (RNN ou GRU), qui produit, à chaque étape t, un vecteur d'état d_t décrivant le contexte courant de la séquence générée ;

— un **mécanisme d'attention**, qui, à chaque étape, calcule un score de compatibilité entre d_t et chaque nœud e_i, permettant de produire une distribution de probabilité sur les nœuds d'entrée.

La sortie du modèle est une séquence d'indices $(y_1, \ldots, y_T)$, où chaque y_t désigne un nœud pointé dans l'entrée.

Adaptation au VRP. En s'inspirant de Nazari et al. (2018), dans notre cas, les entrées sont constituées de trois types d'informations :

— **Statique** : coordonnées (x, y) des clients et du dépôt,

— **Dynamique** : charge restante du véhicule, demande de chaque client.

— **Fixe ou ponctuel** : booléen qui informe si le client est un client ponctuel ou fixe

Ces éléments sont donné à l'encodeur du modèle

Rôle du mécanisme d'attention. Le mécanisme d'attention intervient dans l'actor, à chaque étape du décodage. Son objectif est de calculer une distribution de probabilité sur les clients encore visitables, en tenant compte :

— des informations statiques (positions des clients),

— des informations dynamiques (demandes, charge),

— du contexte séquentiel (état du décodeur RNN).

Plus formellement, à l'étape t, le mécanisme d'attention calcule pour chaque nœud i un score :

$$u_t^i = \vec{v}^\top \tanh(W_1 e_i + W_2 d_t),$$

où e_i est l'embedding du nœud i (vecteur d'encodage), et d_t l'état caché du décodeur. Le softmax appliqué à ces scores donne une distribution sur les actions, permettant à l'agent de pointer vers le prochain client à visiter.
Cette opération est répétée à chaque pas de temps, jusqu'à satisfaction de la demande.

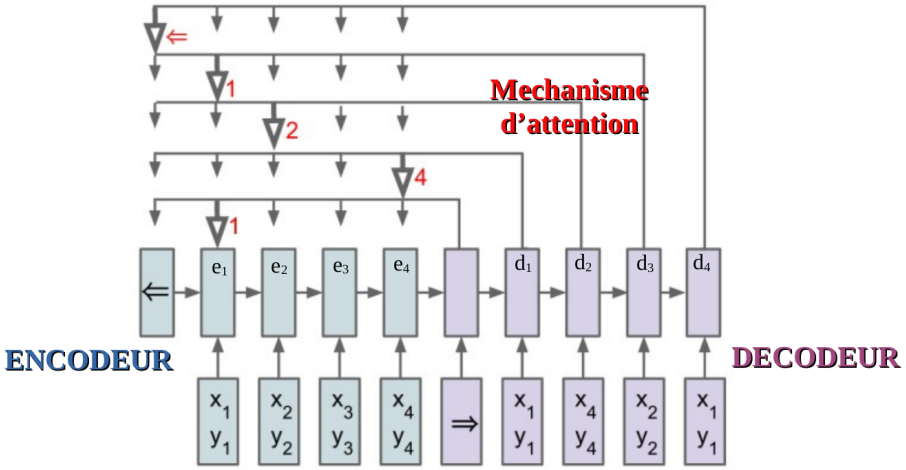


FIGURE 3.2 – Architecture d'un Pointer Network. L'encodeur (en bleu) transforme chaque point (x_i, y_i) en une représentation vectorielle e_i (embedding). Le décodeur (en violet), à chaque étape j, génère un pointeur vers l'un des éléments d'entrée à l'aide d'un mécanisme d'attention. L'illustration a été initialement reprise de Vinyals et al. (2017) que j'ai ensuite modifié.

La Figure 3.2 montre qu'à chaque étape j du décodeur, une distribution de probabilité est calculée sur tous les encodages e_i via le mécanisme d'attention où $\mathbf{d}_j$ est l'état caché du décodeur à l'étape j, et $\mathbf{v}, W_1, W_2$ sont des poids appris. L'application du `softmax` sur les u_j^i donne une distribution sur les éléments encodés, permettant de pointer vers celui à choisir en sortie.
La sortie est une séquence d'indices d'entrée, c'est-à-dire une permutation des (x_i, y_i). Contrairement à un modèle classique qui génère des symboles dans un vocabulaire fixe, le Pointer Network génère des pointeurs vers les entrées. Ici la sortie est la permutation d'indice (1,4,2,1)

3.5 Génération des tournée et améliorations

À partir d'un état initial (dépôt + clients + demandes), l'actor :

— Encode les informations statiques et dynamiques des clients,

— Utilise un décodeur RNN avec mécanisme d'attention pour décider « quel client visiter ».

À chaque étape :

— Le modèle calcule un score u_i pour chaque client accessible,

— Le client ayant le score u_i le plus élevé est directement choisi (mode greedy 3.5),

— Le véhicule passe au client choisi, l'état est mis à jour (demande satisfaite, charge restante ajustée),

— Le processus recommence jusqu'à ce que toutes les demandes soient satisfaites.

Remarque. *Il est courant d'utiliser un mode greedy pour construire les tournées : à chaque étape, le modèle sélectionne l'action la plus probable, c'est-à-dire le nœud associé à la probabilité maximale selon la politique π_θ. Cette approche, déterministe et rapide mais peut conduire à des solutions sous-optimales si le modèle n'a pas suffisamment exploré pendant l'apprentissage.*

Qui prend la décision ? Dans notre modèle, les décisions sont prises par l'**actor**, c'est-à-dire la politique $\pi_\theta(a \mid s)$, qui produit à chaque étape une distribution de probabilité sur les actions possibles. Cette politique est implémentée via une architecture **encodeur–décodeur avec attention**, inspirée des Pointer Networks. Ainsi, l'encodeur-décodeur constitue la structure interne de l'actor, mais c'est bien l'actor dans son ensemble incluant l'attention, le décodeur RNN et les mécanismes de sélection qui génère la séquence d'actions formant la tournée. Ici l'acteur est le PointerNetworks

Communication entre Actor et Critic
Le **critic** intervient en fournissant une estimation de la valeur $V(s_t)$ de l'état courant à chaque étape. Cette estimation permet de calculer un **signal d'avantage** :

$$A(s_t, a_t) = R(\tau) - V(s_t)$$

Il permet d'ajuster la mise à jour de la politique en ne renforçant que les actions réellement bénéfiques. C'est à ce moment que s'opère la communication entre l'actor et le critic : ce dernier influence la direction et l'intensité du gradient appliqué à l'actor.

Amélioration de la politique et du critique
Les paramètres θ de l'actor et ϕ du critique sont mis à jour à l'aide de l'algorithme **REINFORCE** :

$$\theta \leftarrow \theta + \alpha \cdot A(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

où α est le taux d'apprentissage. Chaque décision prise pendant la tournée contribue à ajuster les paramètres θ, qui englobent le poids de l'encodeur, du décodeur et du mécanisme d'attention

Les paramètres ϕ du Critique sont alors mis à jour en minimisant la perte quadratique entre l'estimation $V_\phi(s_t)$ et le retour total $R(\tau)$:

$$\mathcal{L}_{\text{critic}}(\phi) = \left(A(s_t, a_t)\right)^2 = \left(R(\tau) - V_\phi(s_t)\right)^2$$

Enfin, les poids ϕ sont ajustés par descente de gradient sur cette perte :

$$\phi \leftarrow \phi - \beta \cdot \nabla_\phi \mathcal{L}_{\text{critic}}(\phi),$$

où β est le taux d'apprentissage du Critique.

3.6 Adaptation au conVRP

Dans le cadre du Consistent Vehicle Routing Problem (conVRP), une contrainte centrale porte sur la **stabilité des tournées entre plusieurs jours**, en particulier pour les clients dits « fixes », c'est-à-dire ceux que l'on retrouve chaque jour.

Motivation. Dans des contextes réels comme la livraison récurrente de médicaments ou les tournées de soins à domicile, il est souhaitable que :

- les clients fixes soient visités par le même véhicule,
- les clients fixes soient visités dans un ordre similaire,
- les clients fixes soient visités à des horaires stables,
- les livreurs empruntent les même chemins chaque jour

car cela améliore la qualité de service, la satisfaction client et la performance des livreurs. Dans notre cas, on va dans un premier temps, seulement s'intéresser à ce que les livreurs empruntent au maximum les chemins de la veille.

Mesure de la consistance inter-journalière. Afin d'encourager la stabilité des tournées entre jours successifs, nous introduisons une métrique appelée **consistent_scores**. Cette mesure évalue la part des **arcs entre clients fixes** qui sont communs entre deux tournées successives, plus précisément entre celle du jour t (courante) et celle du jour $t-1$ (précédente).

Formellement, pour chaque échantillon i d'un batch (c'est-à-dire une instance de problème VRP), on considère :

— $\mathcal{A}_t^{(i)}$: l'ensemble des arcs (u, v) de la tournée du jour t *entre clients fixes uniquement*

— $\mathcal{A}_{t-1}^{(i)}$: l'ensemble des arcs entre clients fixes dans la tournée du jour $t-1$.

On définit alors le score de consistance de l'échantillon i comme :

$$\text{consistent\_scores}^{(i)} = \frac{|\mathcal{A}_t^{(i)} \cap \mathcal{A}_{t-1}^{(i)}|}{|\mathcal{A}_t^{(i)}|}$$

Ce score est compris entre 0 (aucun arc réutilisé) et 1 (tous les arcs entre clients fixes sont identiques à la veille). La consistance n'est évaluée que sur les arcs entre clients fixes, car ce sont ceux pour lesquels une stabilité est opérationnellement souhaitable (familiarité du livreur, horaires réguliers, etc.).

Formulations de la consistance. Dans le problème *Consistent VRP*, on cherche à favoriser la stabilité des tournées à travers les jours, notamment en conservant les mêmes arcs entre clients fixes.
Deux formulations principales peuvent être utilisées dans la fonction de récompense :

— **Formulation avec bonus :**

$$\text{reward} = -\text{length} + \gamma \cdot \texttt{consistency\_score}$$

Ici, on considère que la consistance vient améliorer la récompense. Le terme $\gamma \cdot$ `consistency_score` agit comme un *bonus* : plus la tournée actuelle est similaire à celle de la veille, plus la récompense est élevée (car le coût total est réduit).

— **Formulation avec pénalité :**

$$\text{reward} = -\text{length} - \gamma \cdot (1 - \texttt{consistency\_score})$$

ou avec pénalité quadratique

$$\text{reward} = -\text{length} - \gamma \cdot (1 - \texttt{consistency\_score})^2$$

Dans cette version, on interprète le manque de consistance comme une *pénalité*. Le terme $(1 - \texttt{consistency\_score})$ représente le manque de consistance, et plus il est grand, plus la pénalité est forte. Inversement, plus la tournée actuelle est similaire à celle de la veille, plus `consistency_score` est proche de 1, ce qui réduit le coût total : la récompense est donc également améliorée.

Dans les deux cas, le paramètre $\gamma > 0$ contrôle l'importance relative accordée à la consistance par rapport à la longueur des tournées. Dans le projet, utiliser la version pénalisé quadratique montrait de bien meilleur résultat.

L'enjeu consiste donc à trouver un compromis pertinent entre la *consistance des tournées* et la *distance totale parcourue*. Il s'agit d'ajuster le paramètre γ de manière à favoriser des trajets stables d'un jour à l'autre, tout en évitant que le modèle n'augmente artificiellement la longueur des tournées pour maximiser la consistance.

3.7 Cadre expérimental

L'objectif est de générer des tournées cohérentes sur plusieurs jours (5 jours), en minimisant à la fois le coût logistique total et la variation des arcs utilisés pour les clients fixes (`consistency_score`).
Dans ce projet, je me suis appuyé sur les hypothèses de stabilité des arcs formulées dans Yao et al. (2021), sur la distinction entre clients fixes et ponctuels introduite par Tarantilis et al. (2012), ainsi que sur l'architecture de type Pointer Networks proposée dans Nazari et al. (2018).

3.7.1 Simulation des données

Soit un VRP à 10 clients, le jeu de données est comme suit :

- Les **clients fixes** occupent 70% du jeu et sont générés une fois pour toutes, aléatoirement dans un carré $[0, 1] \times [0, 1]$, et leur position reste inchangée pour tous les épisodes (jours). Client générés par `torch.rand(1, 2, 7)`
- Les **clients ponctuels** sont générés à chaque épisode, indépendamment, dans la même zone. Client générés par `torch.rand(1, 2, 3)`
- Les **demandes** sont tirées aléatoirement dans un intervalle fixé (ex. entre 1 et 10), et la capacité du véhicule est fixée à une valeur (ex. 30).
- Le dépôt est placé en (0.5,0.5) 3.7.1
- Planning sur 5 jours

Chacun des clients possèdent aussi un flag binaire indiquant si le client est fixe (0) ou ponctuel (1)

Remarque. *Le dépôt a été positionné au centre du carré* $[0, 1]^2$ *afin de refléter le cadre opérationnel d'un problème de tournées cohérentes (conVRP). En effet, dans notre contexte, le planning serait conçu pour une même entreprise, avec un personnel constant, opérant à partir d'un site logistique unique et fixe. Cette hypothèse permet d'évaluer la stabilité des tournées dans un environnement réaliste où le point de départ ne varie pas d'un jour à l'autre.*

Pour l'architecture (clients fixes/ponctuels), on s'est inspiré du travail de Tarantilis et al. (2012). Dans leur papier, les clients ponctuels sont fixés à l'avance, et le jour de leur apparition est connu pour chaque instance. Le planning hebdomadaire est donc entièrement déterministe, ce qui permet d'optimiser les tournées sur un horizon fixe en prenant en compte la structure temporelle exacte. Dans notre approche, en revanche, les clients ponctuels sont générés aléatoirement à chaque jour, indépendamment des jours précédents. Cela rend l'environnement intrinsèquement stochastique : l'agent doit apprendre à

s'adapter à des perturbations quotidiennes imprévisibles, tout en maintenant une certaine stabilité dans les arcs reliant les clients fixes. Cette différence rend notre cadre plus proche des applications réelles, où les demandes ponctuelles varient d'un jour à l'autre.

Modèle utilisé : On rappelle que l'approche retenue repose sur un modèle de type **Pointer Network**, inspiré du travail de Nazari et al. (2018), adapté ici au cadre multi-journalier du ConVRP. Le modèle est entraîné par une méthode de **Policy Gradient**, selon le schéma suivant :

- **Encodeur** : encode les positions et demandes des clients,
- **Décodeur RNN** : génère la tournée séquentiellement, client par client,
- **Mécanisme d'attention** : produit une distribution de probabilité sur les clients restants,
- **Sélection greedy** : à chaque étape, le client avec la probabilité maximale est sélectionné (mode inférence).

Le modèle est entraîné à maximiser la récompense suivante :

$$R(\tau) = -\text{coût total de la tournée} - \gamma \cdot (1 - \texttt{consistency\_score})^2$$

où le second terme pénalise l'incohérence sur les arcs d'un jour à l'autre.

3.7.2 Implémentation et développement

Le projet s'appuie sur le code open source DRL4VRP Nazari et al. (2018) disponible à l'adresse suivante : `https://github.com/mveres01/pytorch-drl4vrp`. Ce code implémente un modèle de type PointerNetworks avec l'architecture actor critic pour résoudre des problèmes de tournées de véhicules (VRP) de manière classique. Il s'agit d'un projet entièrement programmé en Python à l'aide de Pytorch.

Dans le cadre de mon projet, ce code a été adapté pour traiter le conVRP dans laquelle certains clients sont fixes et doivent être visités régulièrement, tandis que d'autres sont ponctuels et varient chaque jour. Une dimension de consistance sur les arcs fixes a été introduite. Les principales modifications sont les suivantes :

Génération des données sur plusieurs jours Une nouvelle classe `ConsistentVRPDataset` a été introduite pour permettre la génération conjointe :

- d'un ensemble de clients **fixes** (mêmes positions et demandes pour tous les jours),
- de clients **ponctuels** différents à chaque jour,
- d'un **planning** s'étalant sur 5 jours.

Cette structure permet de simuler un environnement réaliste de tournées quotidiennes, avec des clients récurrents et d'autres apparaissant aléatoirement chaque jour.

Modification de la fonction de récompense La fonction de récompense a été modifiée en `reward_with_consistency` pour intégrer une notion de **consistance inter-journalière** :

- Les arcs reliant les clients fixes d'un jour à l'autre sont comparés,
- Un **bonus de consistance** a été transformé en **pénalité** proportionnelle à l'incohérence des arcs d'un jour à l'autre,
- Un coefficient `gamma` permet de pondérer cette régularisation.

Scripts d'évaluation et visualisation Plusieurs scripts complémentaires ont été développés pour l'analyse expérimentale :

— `run_gamma.py` : exécute des simulations pour une grille de valeurs du paramètre γ,

— `plot_gamma.py` : génère des visualisations de l'influence de γ sur la longueur des tournées et la stabilité des arcs.

Modifications mineures Le fichier `model.py` n'a été que très peu modifié. En revanche, des ajustements ont été apportés à la fonction d'affichage (`vrp.render`) et certaines modificatiosn dans d'autres fonctions pour compenser mes adaptations :

— Ajout de l'affichage du score de consistance,

— Affichage de la longueur moyenne des tournées.

— Ajout de fonctions, méthodes, lignes de codes

Chapitre 4

Évaluation expérimentale et analyses

Dans cette section, nous présentons les résultats empiriques et des illustrations obtenus à l'aide de notre approche par apprentissage par renforcement pour résoudre le Consistent VRP. J'ai choisi d'explorer une approche par apprentissage, car à ma connaissance, aucune méthode d'apprentissage n'a encore été proposée pour le problème de ConVRP. Ce choix s'explique également par mon intérêt personnel pour ces techniques, ainsi que par l'opportunité qu'elles offrent d'approfondir mes connaissances dans le domaine

4.1 Choix du paramètre de consistance

Le paramètre γ introduit dans la fonction de récompense joue un rôle crucial dans l'équilibre entre l'optimisation logistique (distance totale) et la régularité des tournées (score de consistance). Son choix ne peut être optimisé analytiquement en raison de la nature non supervisée et stochastique du signal de récompense en apprentissage par renforcement. J'ai donc procédé à une recherche empirique sur grille, en évaluant pour chaque valeur de γ les compromis obtenus entre coût et stabilité, notamment via l'analyse de la courbe de Pareto entre les deux critères.

Recherche sur grille (grid search) L'idée est simple : je procède à une exploration empirique du paramètre γ en effectuant une recherche sur grille (boucle `for`). Concrètement, on teste d'abord une plage large de valeurs de γ allant de 1 à 30 avec un pas de 1. Pour chaque valeur, on mesure deux indicateurs : la consistance inter-journalière des tournées et la longueur totale des tournées. Ces résultats sont ensuite visualisés sous forme de courbes permettant d'observer les tendances et d'identifier les zones de compromis possibles entre stabilité et efficacité logistique. Une fois cette première phase exploratoire effectuée, la grille est restreinte à un sous-intervalle plus pertinent, afin de raffiner l'analyse dans la zone de performance optimale.

Courbe de Pareto Dans un problème à objectifs multiples (ou en conflits), comme ici où l'on cherche à minimiser la longueur des tournées tout en maximisant la consistance inter-journalière, il n'existe généralement pas une unique solution optimale. On s'intéresse alors à l'ensemble de Pareto, c'est-à-dire l'ensemble des solutions dites **non dominées**.

Définition : Un point $x = (c_x, \ell_x)$ est **dominé** s'il existe un autre point $y = (c_y, \ell_y)$ tel que :

$$c_y \leq c_x \quad \text{et} \quad \ell_y \leq \ell_x \quad \text{et} \quad (c_y < c_x \quad \text{ou} \quad \ell_y < \ell_x)$$

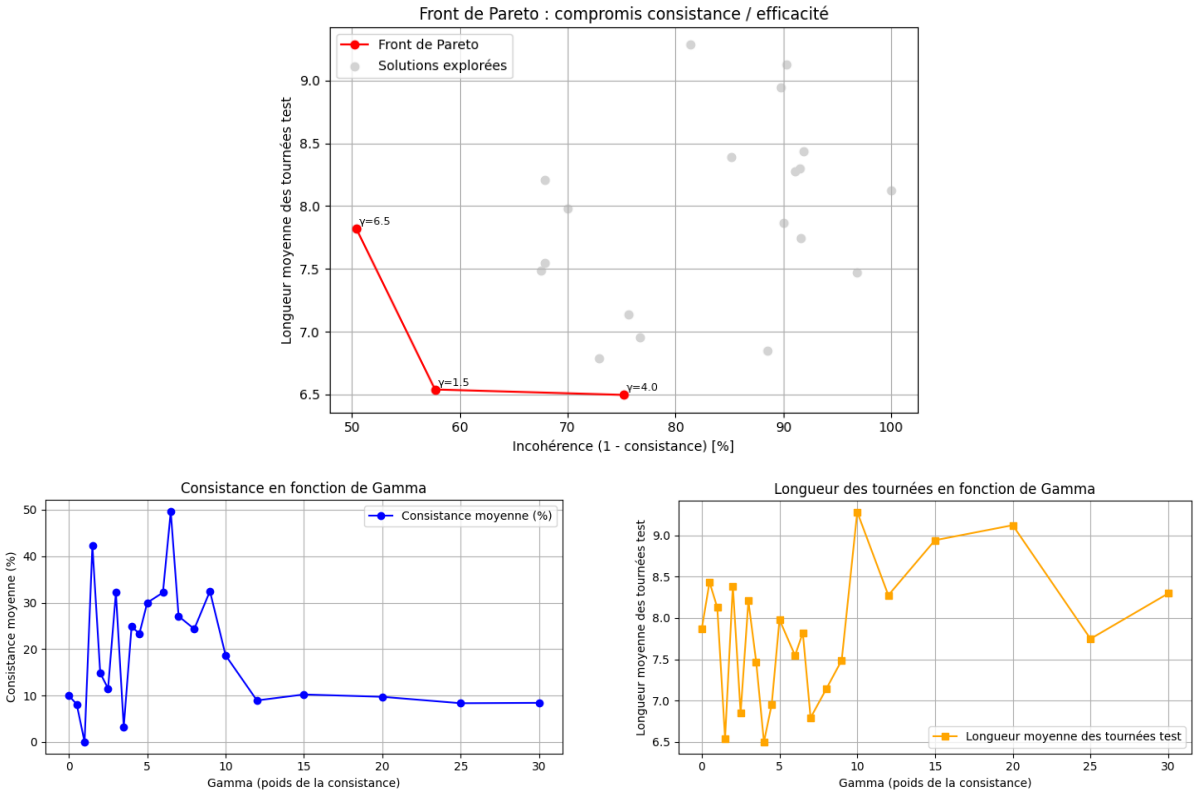
Autrement dit, y est meilleur que x sur tous les objectifs, et strictement meilleur sur au moins l'un d'eux.

En traçant la courbe de Pareto, on identifie visuellement les compromis possibles entre les deux objectifs : toute amélioration de la consistance entraîne généralement une dégradation de la longueur des tournées, et vice versa. Le front de Pareto permet donc de sélectionner une solution "équilibrée" en fonction des priorités.

TABLE 4.1 – Paramètres expérimentaux pour le VRP à 10 clients

Paramètre	**Valeur**
Nombre total de clients	10 (7 fixes, 3 ponctuels)
Capacité maximale du véhicule	35 unités
Demande maximale par client	9 unités
Taille du jeu d'entraînement (par jour)	256 instances
Taille du jeu de validation	64 instances
Taille du batch	12 exemples
Taux d'apprentissage (actor, critic)	5×10^{-4}
Nombre d'epochs	4
Mode d'inférence	Greedy (choix de l'action la plus probable)

(1) Front de Pareto : compromis entre incohérence et longueur de tournée



(a) Consistance moyenne (%) en fonction de γ

(b) Longueur moyenne des tournées en fonction de γ

FIGURE 4.1 – Analyse des performances en fonction de γ : front de Pareto (en haut), et métriques séparées (en bas).

La Figure 4.1 illustre l'effet du paramètre γ sur le compromis entre stabilité des tournées (mesurée par la consistance inter-journalière) et efficacité opérationnelle (mesurée par la longueur moyenne des tournées).

Le graphique du haut représente le **front de Pareto**, en traçant l'incohérence ($1 -$ consistance) en abscisse et la longueur moyenne des tournées test en ordonnée. Chaque point correspond à une valeur de γ, et seuls les points **non dominés** apparaissent en rouge. Ces derniers représentent les meilleurs compromis réalisables entre ces deux objectifs conflictuels. On observe que pour cette réalisation, une valeur modérée de γ (par exemple $\gamma = 4.0$ ou 1.5) permet d'obtenir des tournées relativement courtes tout en maintenant un certain niveau de cohérence (respectivement 30 % et 40 % de consistance). À l'inverse, un γ trop élevé (par exemple $\gamma = 6.5$) améliore la stabilité au détriment d'une augmentation significative de la longueur des tournées, ce qui est généralement peu souhaitable dans un contexte opérationnel.

Les deux graphiques du bas permettent de visualiser séparément l'effet de γ sur chacune des deux métriques. La courbe de gauche montre qu'au-delà d'un certain seuil ($\gamma \geq 10$), la consistance chute brutalement, suggérant que le signal de régularisation devient trop fort et désorganise la politique.

Contrairement à mon intuition , l'augmentation de la consistance ne s'accompagne pas ici nécessairement d'une augmentation significative de la longueur des tournées, et inversement. Autrement dit, on n'observe pas de lien direct ni de compromis net entre stabilité des tournées et efficacité logistique.
En revanche, en analysant les courbes, je remarque une forme de structure récurrente dans les deux graphiques : les variations de la consistance et de la longueur semblent présenter un motif similaire, mais translaté, comme si les effets de γ s'exprimaient avec un décalage selon la métrique observée.

4.2 Impact des contraintes capacitaires

Afin d'étudier l'influence des contraintes capacitaires sur la qualité des tournées produites, nous avons mené une série d'expériences sur des problèmes de VRP avec une capacité maximale fixée à 20 unités par véhicule. Cette valeur, significativement plus restrictive que celle utilisée dans nos expériences précédentes (30 ou 35), limite fortement la quantité de demande qu'un véhicule peut satisfaire avant de devoir retourner au dépôt. En conséquence, les véhicules doivent effectuer davantage d'allers-retours pour satisfaire l'ensemble des clients, ce qui dégrade la performance globale en termes de longueur moyenne des tournées.
En comparaison, avec une capacité de 30 ou 35, le modèle obtenait des tournées significativement plus courtes (de l'ordre de 6.5 à 7.2 en moyenne), tout en maintenant des niveaux de cohérence similaires voire supérieurs 4.1. Cette différence souligne la sensibilité du modèle à la contrainte de capacité : plus celle-ci est sévère, plus la recherche de solutions cohérentes et efficaces devient complexe.
La Figure 4.2 illustre le **front de Pareto** obtenu en faisant varier le coefficient de régularisation γ entre la consistance inter-journalière et la longueur moyenne des tournées. On y observe deux solutions non dominées :

— Une première avec $\gamma = 2.0$, offrant une meilleure stabilité (environ 25 % de cohérence) mais une longueur moyenne des tournées test avoisinant 8.1.

TABLE 4.2 – Paramètres expérimentaux pour le VRP à 10 clients

Paramètre	**Valeur**
Nombre total de clients	10 (7 fixes, 3 ponctuels)
Capacité maximale du véhicule	20 unités
Demande maximale par client	9 unités
Taille du jeu d'entraînement (par jour)	256 instances
Taille du jeu de validation	64 instances
Taille du batch	32 exemples
Taux d'apprentissage (actor, critic)	5×10^{-4}
Nombre d'epochs	4
Mode d'inférence	Greedy (choix de l'action la plus probable)

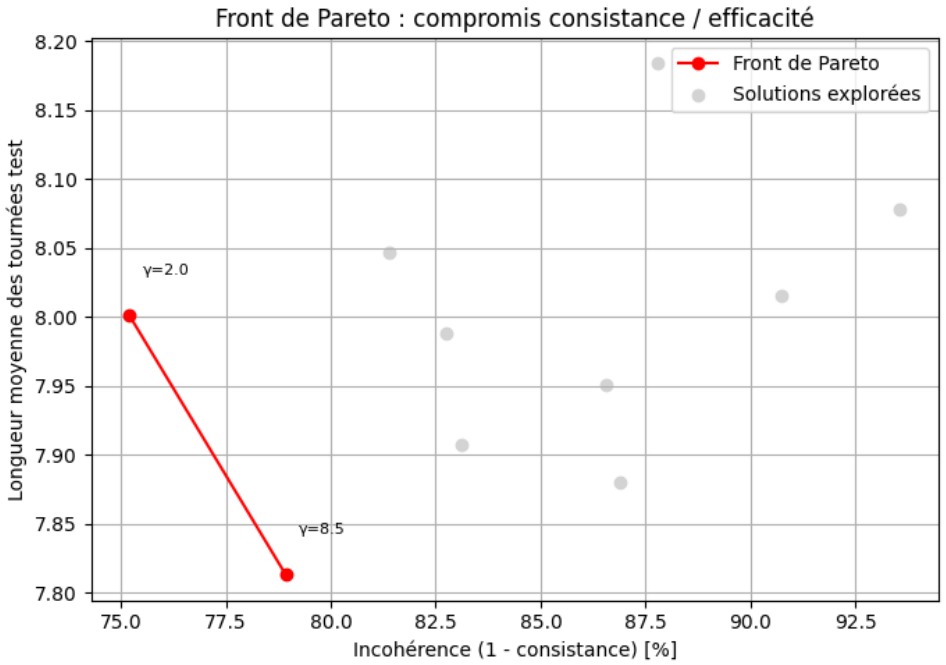


FIGURE 4.2 – Front de Pareto obtenu pour une capacité maximale de 20 unités par véhicule.

— Une seconde avec $\gamma = 0.5$, réduisant la longueur moyenne à environ 7.85 au prix d'une cohérence plus faible (20 %).

En comparaison avec les résultats obtenus à capacité plus élevée, le front de Pareto est ici nettement dégradé : les solutions atteignent des longueurs plus importantes pour un même niveau d'incohérence. Cela met en évidence un compromis beaucoup plus tendu dans le cas de ressources logistiques limitées, rendant la conciliation entre efficacité opérationnelle et régularité des tournées d'autant plus ardue.

4.2.1 Autres résultats

Malgré des configurations similaires, il arrivait fréquemment que la variation du paramètre de consistance γ n'entraîne que peu de changements notables dans les performances. Les solutions obtenues semblaient relativement homogènes, quelle que soit la valeur de la pénalisation. Comme on peut le constater par exemple dans la Figure 4.3. À ce jour, je ne parviens pas à expliquer pleinement ce phénomène.

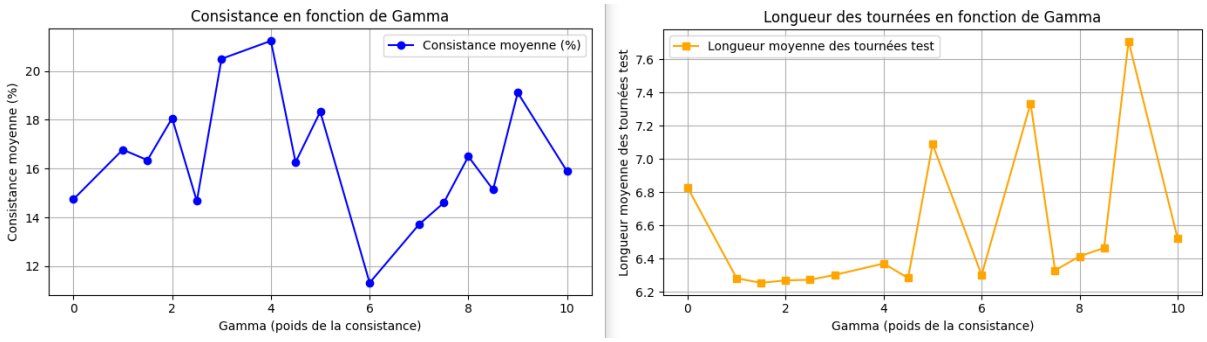
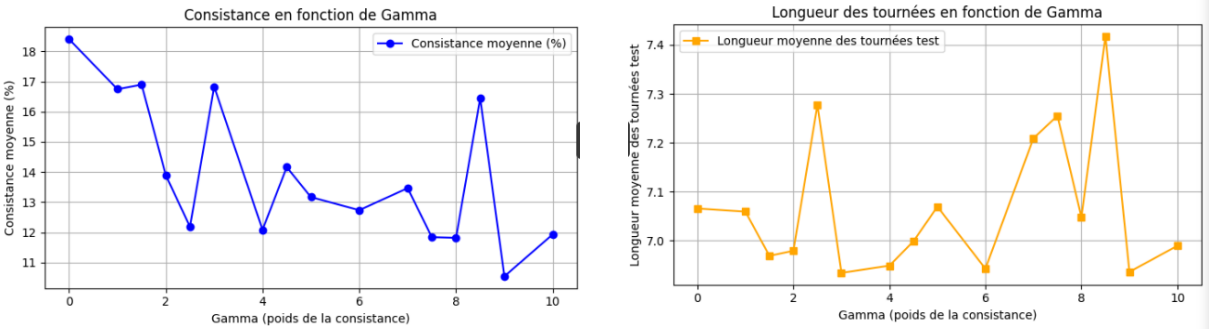


FIGURE 4.3 – Évolution de la consistance moyenne (gauche) et de la longueur moyenne des tournées test (droite) en fonction du paramètre γ

La Figure 4.3 met en lumière un phénomène surprenant : les valeurs élevées de consistance (par exemple autour de $\gamma = 3.5$ ou 4) n'impliquent pas toujours une hausse significative de la longueur moyenne des tournées. Il est compliquer de tirer des conclusions sur ce type de réalisation



(a) Consistance moyenne (%) en fonction de γ

(b) Longueur moyenne des tournées en fonction de γ

FIGURE 4.4 – Impact du paramètre γ sur la consistance des tournées et leur longueur moyenne.

Voici encore un exemple pour la Figure 4.4 où un pattern semble se dessiner avec décalage : la consistance réagit avec un léger retard par rapport à la longueur des tournées.

Plusieurs hypothèses peuvent être avancées pour expliquer cette absence de compromis clair :

— **Plateau de solutions efficaces et stables** : le modèle pourrait atteindre un optimum local dans lequel certaines sous-tournées cohérentes sont naturellement courtes, notamment si les clients fixes sont spatialement proches.

— **Stabilisation indirecte de l'apprentissage** : la régularisation par consistance pourrait jouer un rôle de lissage ou de structuration de la politique apprise, sans forcément dégrader le coût.

— **Masquage par les clients ponctuels** : la variabilité introduite par les clients ponctuels chaque jour pourrait atténuer l'effet attendu de la régularisation, en particulier si les clients fixes sont peu nombreux ou dispersés.

— **Surapprentissage des arcs cohérents** : pour certaines valeurs de γ, le modèle semble réussir à ancrer quelques arcs fixes d'un jour à l'autre, tout en optimisant librement le reste de la tournée.

— **Effet de l'environnement stochastique** : la nature non déterministe du problème, combinée à une taille de batch limitée, pourrait lisser artificiellement certaines variations d'un entraînement à l'autre.

4.3 Illustration du VRP 10

Dans cette section, nous présentons des visualisations qualitatives des tournées générées par le modèle dans le cadre du **VRP avec 10 clients**, dont 7 sont fixes et 3 ponctuels, variant chaque jour. Ces illustrations permettent de mieux comprendre le comportement du modèle, la structure des tournées apprises, ainsi que l'effet du paramètre γ sur la consistance temporelle.

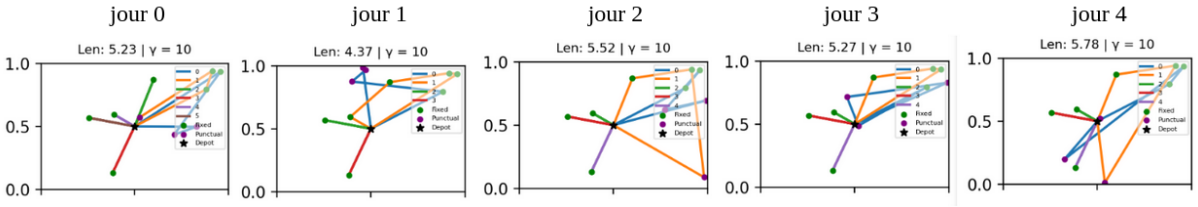


FIGURE 4.5 – Illustration de solutions générées à différents jours, pour un modèle entraîné avec $\gamma = 10$.

La Figure 4.5 présente une séquence de tournées générées à différents jours du planning pour une instance de test fixée. Les clients fixes sont représentés en vert, les ponctuels en violet, le dépôt est une étoile. Chaque couleur correspond à une tournée, c'est à dire un retour au dépôt.
On observe que malgré l'ajout quotidien de clients ponctuels différents, les **arcs entre clients fixes** restent remarquablement similaires dans la structure générale des tournées. Ce phénomène met en évidence l'effet de la régularisation par consistance avec $\gamma = 10$, qui pousse le modèle à **stabiliser certaines portions de trajet** malgré la variabilité introduite jour après jour. Par exemple, plusieurs arcs reliant des clients fixes (points verts) sont présents dans presque toutes les figures, suggérant que le modèle privilégie des sous-tournées partiellement invariantes d'un jour à l'autre.

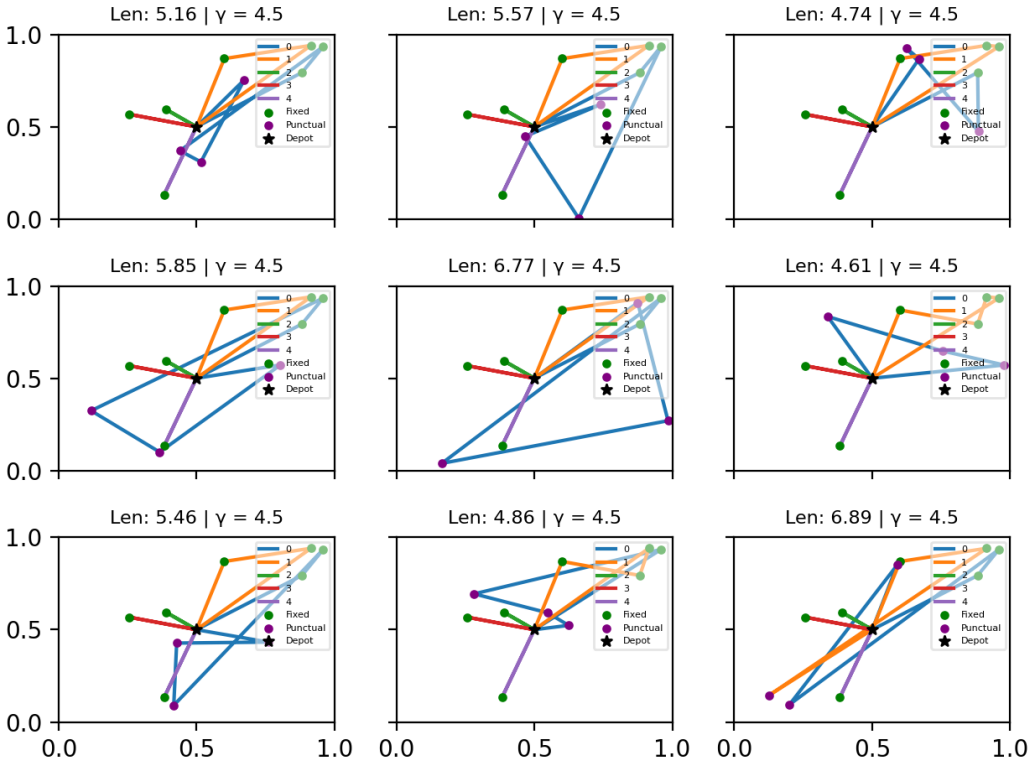


FIGURE 4.6 – Illustration de 9 exemples issus du **jour 4 (test)** pour un modèle entraîné avec $\gamma = 4.5$. Chaque sous-figure correspond à un échantillon d'une même *mini-batch*.

La Figure 4.6 montre neuf tournées générées sur un batch de test lors du **jour 4**, avec une pénalisation de consistance fixée à $\gamma = 4.5$. On observe que, malgré des configurations légèrement différentes entre les échantillons (notamment dans les positions et demandes des clients ponctuels), le modèle semble conserver une structure relativement stable pour les arcs entre clients fixes. Plusieurs motifs d'arcs récurrents apparaissent (par exemple, des chaînes de 2 ou 3 clients fixes parcourus dans le même ordre), traduisant une certaine forme de cohérence intra-batch induite par la régularisation. Toutefois, les arcs m'ont l'air naïf et pourraient surement être optimisés...

Il est important de rappeler ici qu'un batch correspond à un sous-ensemble d'instances VRP traitées simultanément pendant l'entraînement (ou l'évaluation), afin de permettre un calcul efficace du gradient. À chaque itération, un nouveau batch est échantillonné aléatoirement depuis l'ensemble d'entraînement. Le modèle n'est pas entraîné indépendamment sur chaque instance mais optimise sa politique sur l'ensemble du batch via une mise à jour unique des paramètres.

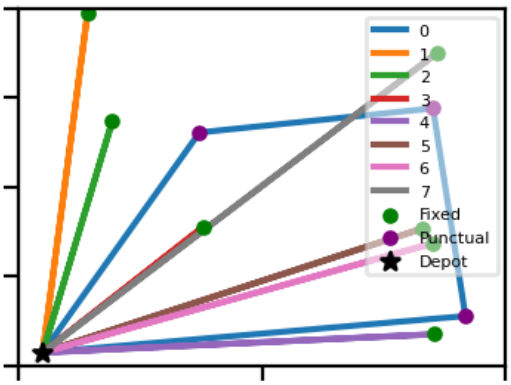


FIGURE 4.7 – Exemple de solution pathologique obtenue avec une valeur trop élevée de γ. Le modèle force tous les arcs à revenir directement au dépôt, nuisant à la qualité des tournées.

La figure a été réalisé grâce à la fonction `render` du projet. Elle représente des tournées sur une grille [0,1]x[0,1]. Ce cas illustre une limite importante de la régularisation par consistance : une valeur de γ trop élevée peut conduire le modèle à détourner l'objectif initial. Dans cet exemple, le modèle génère des tournées artificiellement cohérentes, dans lesquelles chaque client est visité en aller-retour depuis le dépôt, dans le seul but de préserver certains arcs fixes à travers les jours. Ce comportement met en évidence la nécessité de calibrer finement le poids de la pénalité pour ne pas compromettre l'efficacité globale des tournées.

Chapitre 5

Limites du travail et perspectives

Diificultés recontrées L'une des principales difficultés rencontrées au cours de ce travail a été l'obtention d'un code fonctionnel, stable et suffisamment représentatif de la réalité opérationnelle visée. L'adaptation du code d'apprentissage par renforcement au cas particulier du conVRP avec contraintes de consistance inter-journalière m'a demandé un travail conséquent d'ingénierie et de vérification.

Une fois cette étape franchie, un temps important a été consacré à comprendre le rôle des hyperparamètres, en particulier leur impact sur la stabilité de l'apprentissage et sur les compromis entre performance et consistance. L'analyse des signaux d'apprentissage (récompenses, avantages, scores de consistance) s'est révélée délicate, en particulier dans un environnement stochastique où les résultats peuvent fluctuer fortement entre deux exécutions, même avec une graine fixée.

Une autre difficulté majeure a été l'absence de benchmark ou papier correspondant exactement à mon cadre : tournées planifiées sur plusieurs jours, clients ponctuels renouvelés chaque jour, et notion explicite de consistance des arcs. Cela a limité les possibilités de comparaison directe avec d'autres méthodes existantes.

Enfin, l'interprétation des résultats expérimentaux a souvent soulevé des questions méthodologiques : certaines modifications censées améliorer les performances ont au contraire empiré les résultats, imposant de comprendre les causes, de diagnostiquer finement les effets, puis de revenir en arrière ou de corriger le protocole.

Ces limites soulignent à la fois la complexité du problème étudié et les défis liés à son évaluation. Elles ouvrent naturellement la voie à plusieurs perspectives que nous détaillons ci-après.

Perspectives Par manque de temps, plusieurs pistes n'ont pas pu être explorées en profondeur, mais ce projet me semble loin d'être clos, même après la soumission du mémoire. À court terme, une priorité sera de comparer mon approche à des benchmarks existants, comme ceux proposés par Groër et al. (2009) ou d'autres méthodes robustes de la littérature. Une adaptation du cadre expérimental pourrait être envisagée pour rapprocher le problème étudié de ceux abordés dans ces références, par exemple en générant les clients ponctuels de manière connue à l'avance pour intégrer une planification anticipée, à la manière de Tarantilis et al. (2012).
D'autres perspectives incluent l'étude plus fine de l'influence des hyperparamètres. Certaines modifications, telles qu'une augmentation du `batch_size`, ont eu un impact négatif

sur les performances, sans explication claire à ce jour. Un travail plus approfondi sera nécessaire pour comprendre les interactions entre la taille de l'échantillon d'entraînement (`train_size`), le taux d'apprentissage, la taille de lot, ainsi que la proportion de clients fixes dans chaque instance.

Enfin, des extensions méthodologiques sont envisagés : il serait intéressant de tester des heuristiques classiques (comme savings, Clarke-Wright ou des méthodes de regroupement) adaptées à ce cadre stochastique multi-jour, ou d'exploiter des solveurs existants comme Google OR-Tools pour valider certains cas ou générer des solutions de référence.

Une autre direction naturelle pour améliorer la pertinence opérationnelle du modèle est l'ajout de contraintes supplémentaires. En particulier, l'introduction de fenêtres temporelles permettrait de modéliser des situations où chaque client doit être visité dans un intervalle horaire spécifique, ce qui est fréquent dans des contextes réels comme la livraison en pharmacie ou les soins à domicile. Cette contrainte impose non seulement un ordonnancement des visites cohérent avec la disponibilité des clients, mais elle peut également affecter fortement la structure des tournées optimales.

Enfin, Au-delà de la simulation, j'aimerais également, à l'avenir, tester mon approche sur un jeu de données réel représentant une ville avec de vraies demandes. Deux pistes s'offrent pour cela. D'une part, l'utilisation d'une API comme Google Maps permettrait de récupérer des distances réalistes entre adresses, en tenant compte du réseau routier, des temps de trajet et éventuellement du trafic. D'autre part, il serait intéressant d'exploiter un jeu de données public contenant des demandes journalières, issues par exemple de tournées de soins à domicile ou de logistique urbaine, comme dans certaines études antérieures Yao et al. (2021).

Conclusion

L'objectif de ce TER était d'étudier le *Consistent Vehicle Routing Problem* (ConVRP) dans un cadre *stochastique et multi-jour*, en explorant une approche basée sur l'apprentissage par renforcement profond à l'aide de *Pointer Networks*. Ce projet, bien que complexe à plusieurs niveaux, s'est révélé particulièrement formateur et stimulant.

La mise en œuvre technique a d'abord constitué un défi majeur. De l'adaptation fine du code aux spécificités du problème à la stabilisation des entraînements dans un environnement fortement aléatoire, chaque étape a exigé des efforts d'adaptation, de compréhension et de révision. Pourtant, ces difficultés n'ont pas été des obstacles vains : elles m'ont permis de me confronter à la réalité concrète du travail de recherche, avec ses moments de doute, d'impasses, mais aussi ses percées enthousiasmantes.

Cette expérience m'a ainsi offert une première immersion dans la démarche scientifique : identifier une problématique, formaliser un cadre expérimental pertinent, tester des hypothèses, interpréter des résultats parfois inattendus, proposer des ajustements. J'ai découvert le plaisir de m'approprier un sujet initialement inconnu, de plonger dans la littérature, de coder, de déboguer, d'itérer et, parfois, d'avoir la satisfaction de voir une idée fonctionner.

Au terme de cette exploration, nous avons notamment pu analyser l'impact des **contraintes capacitaires** sur les performances du modèle, ainsi que l'effet du **paramètre de consistance** sur la stabilité des arcs inter-journaliers.
Bien que ce TER ne puisse résumer la richesse d'un travail doctoral, il m'a permis de mesurer l'exigence, la patience et la curiosité intellectuelle que nécessite tout projet de recherche. Le cheminement, plus que le résultat final, a constitué la véritable valeur ajoutée de ce travail.

Je ne considère pas ce projet comme achevé. Plusieurs pistes évoquées comme la comparaison avec des benchmarks existants, l'intégration de contraintes plus réalistes comme les fenêtres de temps, des test sur données réelles, etc. restent ouvertes et mériteraient d'être explorées. Ce mémoire constitue ainsi, à mes yeux, une première étape et non une finalité.

Enfin, ce projet m'a permis de prendre un recul précieux sur la méthodologie de recherche. Je comprends désormais l'importance d'une planification structurée dès le départ, d'un questionnement clair. Les erreurs que j'ai pu commettre dans l'ordre des priorités, dans l'évaluation de certains signaux d'apprentissage, ou dans mes choix de paramètres m'ont enseigné des leçons durables. À l'avenir, je me sens plus à même d'aborder un projet de recherche avec méthode, rigueur, et confiance

Bibliographie

Bono, G. (2020). *Deep multi-agent reinforcement learning for dynamic and stochastic vehicle routing problems*. Thèse de doctorat, Université de Lyon, France. NNT : 2020LYSEI096, https://theses.hal.science/tel-03098433.

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1) :80–91.

Glover, F. and Laguna, M. (1999). *Tabu search I*, volume 1.

Groër, C., Golden, B., and Wasil, E. (2009). The consistent vehicle routing problem. *Manufacturing & Service Operations Management*, 11(4) :630–643.

Haugland, D., Ho, S. C., and Laporte, G. (2007). Designing delivery districts for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, 180(3) :997–1010.

Lamprier, S. (2019). Introduction à l'apprentissage par renforcement. https://dac.lip6.fr/wp-content/uploads/2019/10/coursRLD4.pdf. Cours DAC SorbonneUniversite, 2019.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097–1100.

Nazari, M., Oroojlooy, A., Snyder, L. V., and Takáč, M. (2018). Reinforcement learning for solving the vehicle routing problem.

Shahbazian, R., Pugliese, L. D. P., Guerriero, F., and Macrina, G. (2024). Integrating machine learning into vehicle routing problem : Methods and applications. *IEEE Access*, 12 :93087–93115.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2) :254–265.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks.

Tarantilis, C., Stavropoulou, F., and Repoussis, P. (2012). A template-based tabu search algorithm for the consistent vehicle routing problem. *Expert Systems with Applications*, 39(4) :4233–4239.

Toth, P. and Vigo, D. (2014). *Vehicle routing : problems, methods, and applications*. SIAM.

Vinyals, O., Fortunato, M., and Jaitly, N. (2017). Pointer networks.

Weng, L. (2022). Deep reinforcement learning : Actor critic. https://www.youtube.com/watch?v=oDdPcEanLwY.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4) :229–256.

Yao, Y., Van Woensel, T., Veelenturf, L. P., and Mo, P. (2021). The consistent vehicle routing problem considering path consistency in a road network. *Transportation Research Part B : Methodological*, 153 :21–44.