VRP – VEHICLE ROUTING PROBLEM

Membres du projet	Encadrant du projet
CISERANE Marius BOULLOT Matthias	BONNEVAY Stéphane

Réalisé le 11 mai 2025

Sommaire

Table des figures et tableaux

Introduction

Contexte et enjeux du VRP

Le Vehicle Routing Problem (VRP) est un problème d'optimisation combinatoire qui se pose ainsi :

« Quel est l'ensemble optimal d'itinéraires qu'une flotte de véhicules doit emprunter pour livrer un ensemble donné de clients ? »

Apparu pour la première fois dans l'article de George Dantzig et John Ramser en 1959 – où la première approche algorithmique a été appliquée aux livraisons d'essence – il constitue une généralisation du célèbre Traveling Salesman Problem (TSP). Dans sa forme la plus répandue, on part d'un dépôt central où est stockée la marchandise et on doit desservir plusieurs clients ayant passé commande.

Les enjeux industriels sont majeurs :

- **Réduction des coûts logistiques** : les fournisseurs d'outils VRP annoncent souvent des économies de 5 à 30 % sur les tournées de livraison.
- **Impact environnemental** : diminuer la distance parcourue limite la consommation de carburant et les émissions de CO_2.
- **Complexité algorithmique** : le VRP est NP-difficile ; les méthodes exactes (programmation mathématique, branch-and-bound) ne traitent que de petits cas. En pratique, on recourt à des heuristiques et méta-heuristiques pour résoudre les instances de taille réelle en temps raisonnable.

Au croisement de la recherche opérationnelle et des besoins métiers, le VRP est aujourd'hui au cœur des solutions de planification de tournées dans le e-commerce, la grande distribution et bien d'autres secteurs.

Problématique et objectifs

La variante précise que nous abordons dans notre projet est la suivante : Capacitated Vehicle Routing Problem with Time Windows (CVRPTW)

Problématique

Minimiser la distance totale sous contraintes de capacité des véhicules et de fenêtres horaires de service chez les clients.

Objectif principal

Créer un logiciel capable d'optimiser la logistique de livraison d'un dépôt de marchandises, en générant des tournées réalistes et proches de l'optimum.

Objectifs secondaires

- Réduire le temps de calcul pour rendre l'outil exploitable en conditions opérationnelles.

- Améliorer la robustesse et la répétabilité des solutions (faible variabilité entre exécutions).
- Concevoir une architecture modulaire pour pouvoir intégrer ultérieurement de nouvelles contraintes (multi-dépôts, priorités clients…).

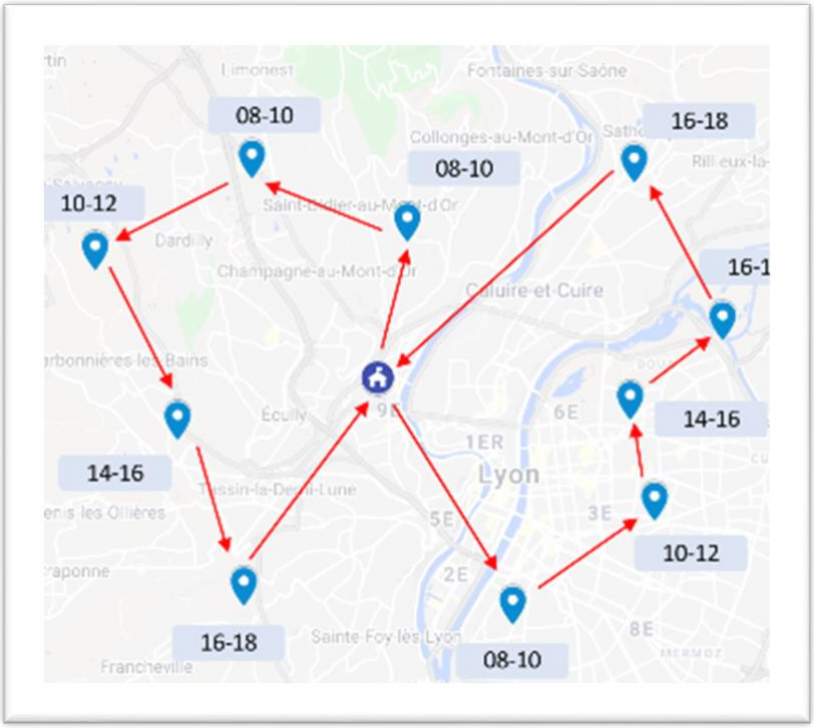


Figure 1 : Représentation visuelle du problème CVRPTW

Section 1 : Méthodologie

État de l'art des algorithmes VRP

Les algorithmes métaheuristiques sont aujourd'hui incontournables pour résoudre des instances de VRP de taille réelle. Parmi les plus répandus, on distingue :

- **Algorithmes génétiques**
  - Inspirés de la sélection naturelle, ils maintiennent une population de solutions (tournées) et appliquent tour à tour des opérateurs de croisement et de mutation.
  - Après évaluation de chaque individu par une fonction de coût (distance, capacité, fenêtres), les meilleures solutions sont sélectionnées pour former la génération suivante.
- **Méthode Tabou**
  - Basée sur une descente locale enrichie d'une mémoire de courts termes (« liste tabou ») qui interdit temporairement certains mouvements pour sortir des minima locaux.
  - On accepte parfois des solutions moins bonnes pour explorer de nouveaux voisins, avant de revenir à la descente stricte.
- **Recuit simulé**
  - Commence par accepter des mouvements vers des solutions de coût plus élevé avec une probabilité décroissante (fonction de la « température »).
  - Au fur et à mesure que la température baisse, le comportement se rapproche d'une recherche locale stricte, stabilisant la solution.
- **Recherche adaptative dans de grands voisinages (ALNS)**
  - Combine dynamiquement plusieurs opérateurs de destruction et de reconstruction (ex. relocate, exchange, 2-opt).
  - Les opérateurs les plus performants sont progressivement favorisés grâce à un mécanisme de score adaptatif.

Approche adoptée

Pour ce projet, nous avons retenu un **algorithme génétique classique**, appliqué de la façon suivante :

1. **Initialisation**
   a. Construction aléatoire d'une population de solutions respectant les contraintes de capacité et de fenêtres horaires.
2. **Opérateurs génétiques**
   a. **Sélection** : on conserve à chaque itération les solutions les plus performantes (évaluation par distance totale).
   b. **Croisement & mutation** : application d'opérateurs simples (relocate, exchange, 2-opt, etc.) pour générer de nouveaux individus.
3. **Critère d'arrêt**

a. La boucle principale s'interrompt dès que la meilleure solution ne s'améliore plus ou qu'un seuil de temps fixé est atteint.

Flux de données et interface

- Les instances sont importées/exportées au format textuel « .vrp ».
- L'interface utilisateur est déployée dans un navigateur web, communicant avec le moteur Python via des requêtes HTTP, pour garantir portabilité et facilité d'utilisation.

Répartitions des tâches

Le diagramme de Gantt associe chaque lot de travail aux membres de l'équipe :

- **Planification du projet**
- **Extraction des données & création des objets**
- **Implémentation des opérateurs simples**
- **Développement de la boucle principale de l'algorithme génétique**
- **Encodage des résultats & développement de l'interface utilisateur**
- **Ajout des opérateurs complexes & ajustements de l'interface**
- **Gestion de la contrainte de temps & tests d'intégration**

Rôles et responsabilités

- **Marius** : Chef de projet & Développeur back-end
- **Matthias** : Administrateur systèmes & Développeur front-end

Des tests unitaires et fonctionnels ont été réalisés de manière continue tout au long du développement pour garantir la qualité et la robustesse de la solution.

Voir l'annexe 1 pour le diagramme de Gantt

Section 2 : Développement de l'application

Implémentation des algorithmes

Opérateur Relocate

- **Principe** : on retire un client d'une tournée et on l'insère à une position différente, soit dans la même tournée, soit dans une autre.
- **Effet** : modifie la charge et la distance de deux tournées à la fois, permet des ajustements fins sans casser la structure générale.

Opérateur Exchange

- **Principe** : on échange la position de deux clients, chacun restant dans sa tournée d'origine (mais pas obligatoirement).
- **Effet** : peut redistribuer la charge et améliorer l'équilibre entre tournées, tout en modifiant légèrement la distance totale.

Opérateur Cross-Exchange

- **Principe** : on sélectionne deux segments de tournées (suite de clients) dans deux tournées différentes, puis on les échange intégralement.
- **Effet** : changements plus drastiques, potentiellement meilleurs gains sur la distance mais risque de violer des contraintes si mal paramétrées.

Opérateur 2-Opt

- **Principe** : dans une même tournée, on supprime deux arêtes et on reconnecte les clients de façon croisée, inversant ainsi le sous-chemin intermédiaire.
- **Effet** : amélioration rapide des tournées en supprimant les croisements, très efficace pour la réduction locale de la distance.

Voir l'annexe 2 pour une représentation visuelle

Conception des interfaces

Afin d'interagir avec l'algorithme d'optimisation, il faut implémenter une ou plusieurs interfaces pour les utilisateurs, mais également les développeurs.

Représentation des données

Ces interfaces doivent en particulier être capable de transformer la représentation des données entrantes afin qu'elles soient compréhensibles avec l'algorithme, et de transformer les données sortantes afin qu'elles soient compréhensibles par l'utilisateur.
La représentation des données côté utilisateur se fait avec le format de fichiers en ".vrp". Ces fichiers encodent la donnée des trajets et des clients, un premier algorithme se charge de *parser* sont contenu pour qu'il puisse être traité par le programme, et un autre encode les données sortantes.

Interface développeur

Les développeurs ont accès à une interface spécialisée, qui a pour avantage de ne pas faire interfacer le serveur entre le programmeur et son code. Deux affichages ont été programmées : Une première interface sobre s'utilisant depuis la console, permet de rapidement exécuter les algorithmes. Une seconde interface, cette fois graphique, donne un aperçu visuel de l'exécution de l'algorithme, en affichant en direct les trajets au fur et à mesure que les opérateurs les modifient.

Interface utilisateur

Les utilisateurs interagissent avec l'application par le biais de leur navigateur. Les différentes requêtes sont envoyées par le navigateur au serveur python, qui les traite de façon asynchrone. L'interface est constituée en 3 étapes :

- Accueil : sur l'accueil, l'utilisateur peut sélectionner un ensemble de fichiers .vrp à optimiser. L'interface lui permet de sélectionner des fichiers en plusieurs fois, ainsi que de supprimer ceux ajoutés par erreur. L'utilisateur peut alors décider de soumettre l'ensemble de ses fichiers au serveur, qui démarrera alors le processus d'optimisation et amènera l'utilisateur sur la page de chargement.
- Traitement : Lors de la phase de traitement, l'utilisateur a un visuel sur l'ensemble des fichiers en cours de traitement et leur état : en cours / terminé avec succès / abandon de l'opération. Un système de Web Socket permet au serveur de notifier en temps réel l'utilisateur lorsque le traitement d'un fichier est terminé ou a été abandonné. Lorsque tous les traitements sont finis, le Web Socket est fermé, et le serveur redirige l'utilisateur sur la page des résultats.
- Résultats : À partir de cette page, l'utilisateur peut télécharger les fichiers traités individuellement ou bien tous à la fois sous format .zip, et à la possibilité de revenir à l'accueil pour répéter l'opération.
  Pendant toutes les étapes, l'utilisateur a à sa disposition une page d'aide pour s'orienter.

Le serveur web est instancié par le code python, et tourne en local sur la machine de l'utilisateur, permettant une connexion sans réseau.

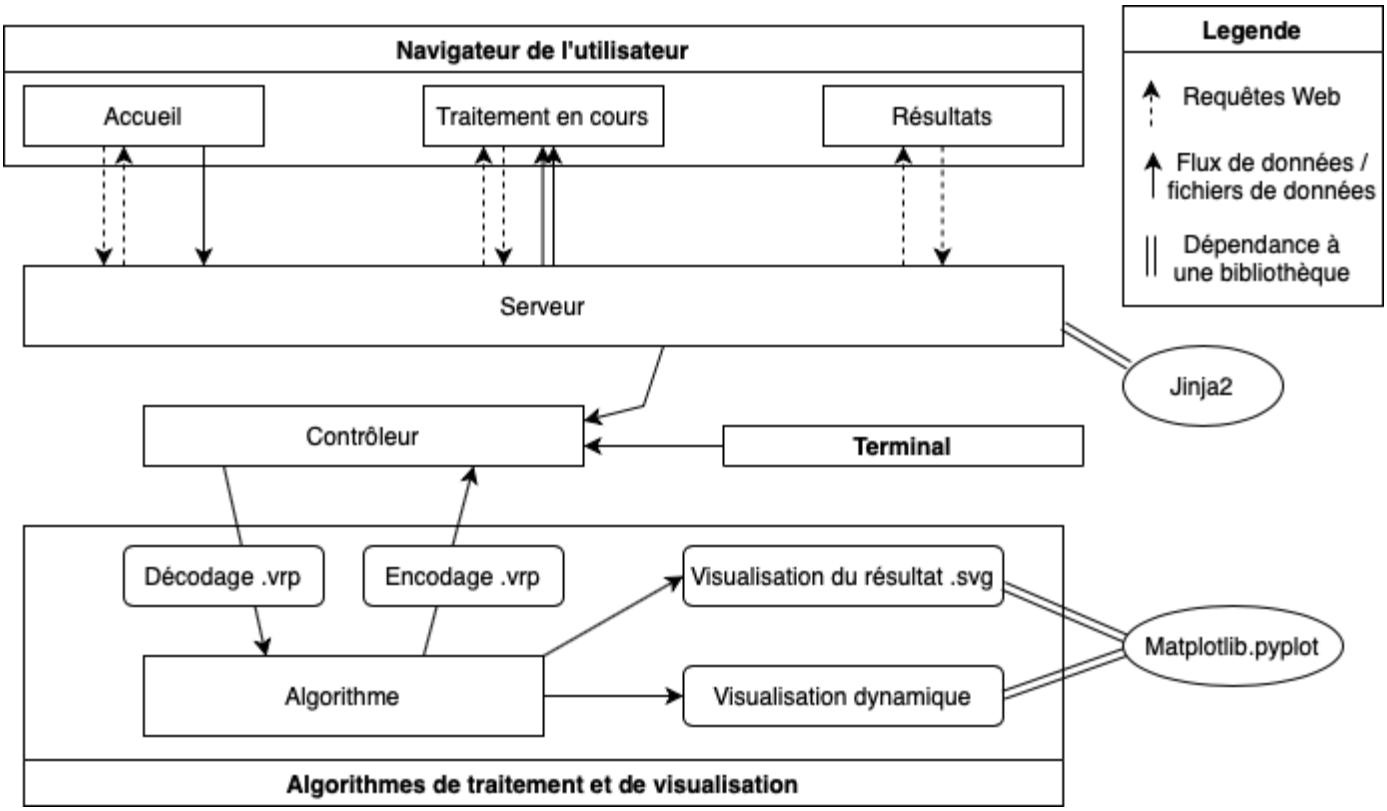


Figure 2 : Diagramme de la communication et du flux de données de l'application

Tests

Afin de nous assurer du bon fonctionnement des algorithmes, nous avons mis en place une série de tests se décomposant en deux niveaux :
Les tests statiques, sont réexécuté à chaque modification du code :

- Les tests unitaires sont réalisés sur les routines constituant le cœur de l'algorithme car ce sont par conséquent les passages les plus empruntés par le programme.
- Les tests d'intégrations s'assurent que les fonctions plus compliquées à tester exhaustivement, comme celles d'import et d'export des données.

Les tests dynamiques, sont exécutés en même temps que les algorithmes :

- Des assertions sont présentes tout le long de l'exécution du programme, afin de confirmer que les transformations successives des données n'invalident aucune présupposition.
- Un test à la fin de l'exécution permet de vérifier que les contraintes temporelles et de capacités ont bien été respectées.

Section 3 : Résultats

Méthodes d'analyses

Afin d'évaluer différentes métriques, nous avons générer des fichiers de données aléatoire et non-optimisés pour 10, 20, ... jusqu'à 210 clients (soit 21 fichiers). Les nombres et intervalles ayant été choisis pour garder les temps de calculs raisonnables. Puis chaque fichier a été optimisé par l'algorithme suivant différentes situations et métriques. En particulier, nous avons testé quatre situations :

1. une première avec uniquement l'opérateur relocate,
2. une seconde avec les opérateurs relocate et exchange,
3. une troisième en rajoutant l'opérateur cross-exchange, et
4. une dernière avec tous les opérateurs.

Pour chacune de ces situations, nous avons testé une métrique sans contrainte, une avec contrainte de capacité, et une dernière avec contrainte de capacité et de temps.

Analyse quantitative

Grâce aux données acquises, voici ci-dessous les résultats obtenus.

Tableau 1 : Performances de l'algorithme sans contraintes

Situation	Gain (en %)	Nombre d'itérations par clients	Temps d'exécution (en secondes)
1	84,96	0,9789	19,73
2	84,96	0,9805	19,63
3	87,90	1,215	21,89
4	88,16	1,225	21,41

Tableau 2 : Performances avec une contrainte de capacité

Situation	Gain (en %)	Nombre d'itérations par clients	Temps d'exécution (en secondes)
1	84,39	0,9901	15,34
2	84,39	0,9896	16,00
3	85,55	1,084	17,26
4	85,64	1,088	20,68

Tableau 3 : Performances avec contrainte de capacité et d'intervalles de temps

Situation	Gain (en %)	Nombre d'itérations par clients	Temps d'exécution (en secondes)
1	64,90	1,063	24,65
2	65,11	1,062	25,275
3	67,40	1,019	31,30
4	67,47	1,006	29,83

L'analyse des performances de l'algorithme d'optimisation de trajet révèle des résultats significatifs, indépendamment des situations testées. En effet, les gains observés sont substantiels, avec une réduction de la distance parcourue d'environ 84,96 % dans les scénarios sans contraintes, et d'environ 67,47 % lorsque toutes les contraintes sont appliquées.

L'ajout d'opérateurs à l'algorithme d'optimisation a permis d'améliorer les gains de 2 à 5 % en moyenne. Cette amélioration peut être expliquée par le fait que l'algorithme s'arrête à la première solution optimale locale rencontrée. Ainsi, dans certaines situations, un opérateur peut ne pas être en mesure d'identifier une solution plus avantageuse, tandis qu'un autre opérateur pourrait le faire. Par conséquent, la diversité des opérateurs contribue à une exploration plus exhaustive de l'espace de solutions.

Il est également crucial de noter que l'introduction de contraintes a un impact négatif sur les gains réalisés. Les résultats montrent une diminution des gains, passant d'environ 86,5 % dans le scénario sans contrainte à environ 85 % avec une contrainte de capacité, et à environ 65 % lorsque des contraintes de capacité et de temps sont ajoutées. Cette réduction des gains peut être attribuée à la restriction du nombre de solutions optimales disponibles, ce qui diminue la probabilité de les identifier.

Le graphique présenté en Figure 3 illustre les performances de l'algorithme en fonction du nombre de clients à livrer. Comme anticipé, le temps d'exécution de l'algorithme suit une croissance quadratique, notée $O(n^2)$, ce qui est cohérent avec la complexité algorithmique attendue pour le programme implémenté.

De manière surprenante, nous observons que le nombre de clients est proportionnel au nombre d'itérations effectuées par l'algorithme. Cette relation inattendue mérite une exploration plus approfondie, car elle pourrait indiquer des dynamiques sous-jacentes dans le processus d'optimisation qui ne sont pas immédiatement évidentes.

En ce qui concerne les gains réalisés, il était prévisible que ceux-ci tendent vers une limite à mesure que le nombre de clients augmente. En l'occurrence, le graphique nous informe que le gain tend vers les 85%.

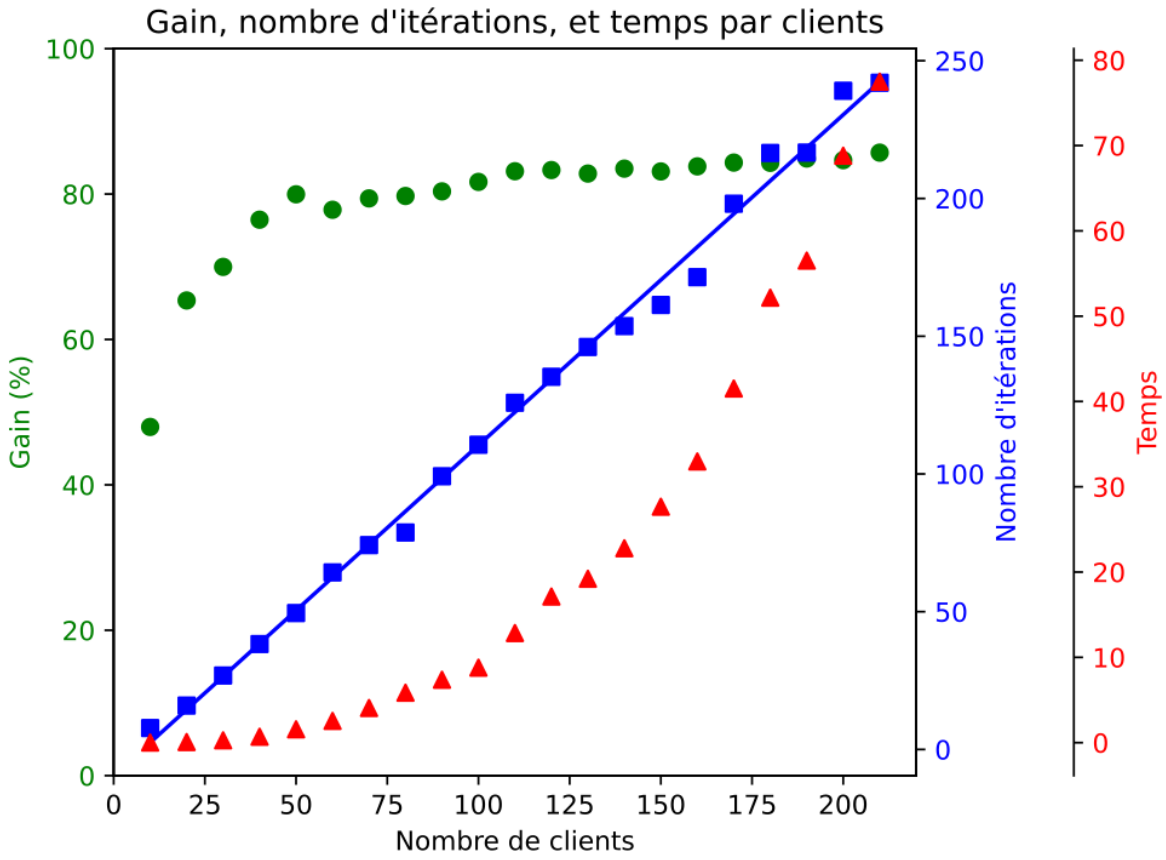


Figure 3 : Performances selon le nombre de clients à livrer

Section 4 : Bilan critique

Succès et apport

Dans son ensemble, la réalisation du projet a été mené à bien. L'application est pleinement fonctionnelle autant par son algorithme d'optimisation efficace, que par son interface d'utilisation aboutie.
Le travail au long du projet a été effectué efficacement, en grande partie grâce à la répartition du travail bien partagée. Le diagramme de Gantt, dans sa première moitié, a été respecté dans sa totalité autant du point de vue des tâches à réaliser que des intervalles de temps. En revanche, la deuxième partie portant sur l'implémentation de l'algorithme Tabou ou Recuit Simulé n'a pu être complétée dû à un manque de temps durant la période concernée.
Au cours de ce projet, nous avons remarqué qu'une bonne communication était primordiale, sans laquelle il aurait été presque impossible de se coordonner et ainsi tenir les délais.
De plus, ce projet a également été un support éducatif en grande partie pour ce qui est des algorithmes méta-heuristiques, que l'on a trouvé facilement abordable malgré nos attendus.

Limites et difficultés

Pendant le développement des algorithmes, nous avons fait face à de multiples reprises à un bug, dont on suspecte encore la présence dans le code final, malgré des tests multiples. Le problème en question intervient *principalement* lorsqu'un opérateur tente une opération qui ne change pas concrètement un trajet : comme par exemple, échanger l'ordre d'un client avec un lui-même. Dans ce cas-là les erreurs d'imprécision des nombres flottant peuvent rendre le gain de l'opération non nul, mais néanmoins ridiculement petit. On se retrouve alors dans une boucle infinie, ou une opération est exécutée en continue sans jamais modifier les trajets. Le problème fut partiellement résolu en évitant les cas cis-mentionné sur l'ensemble des opérateurs.

Conclusion et perspectives

Synthèse des apports majeurs

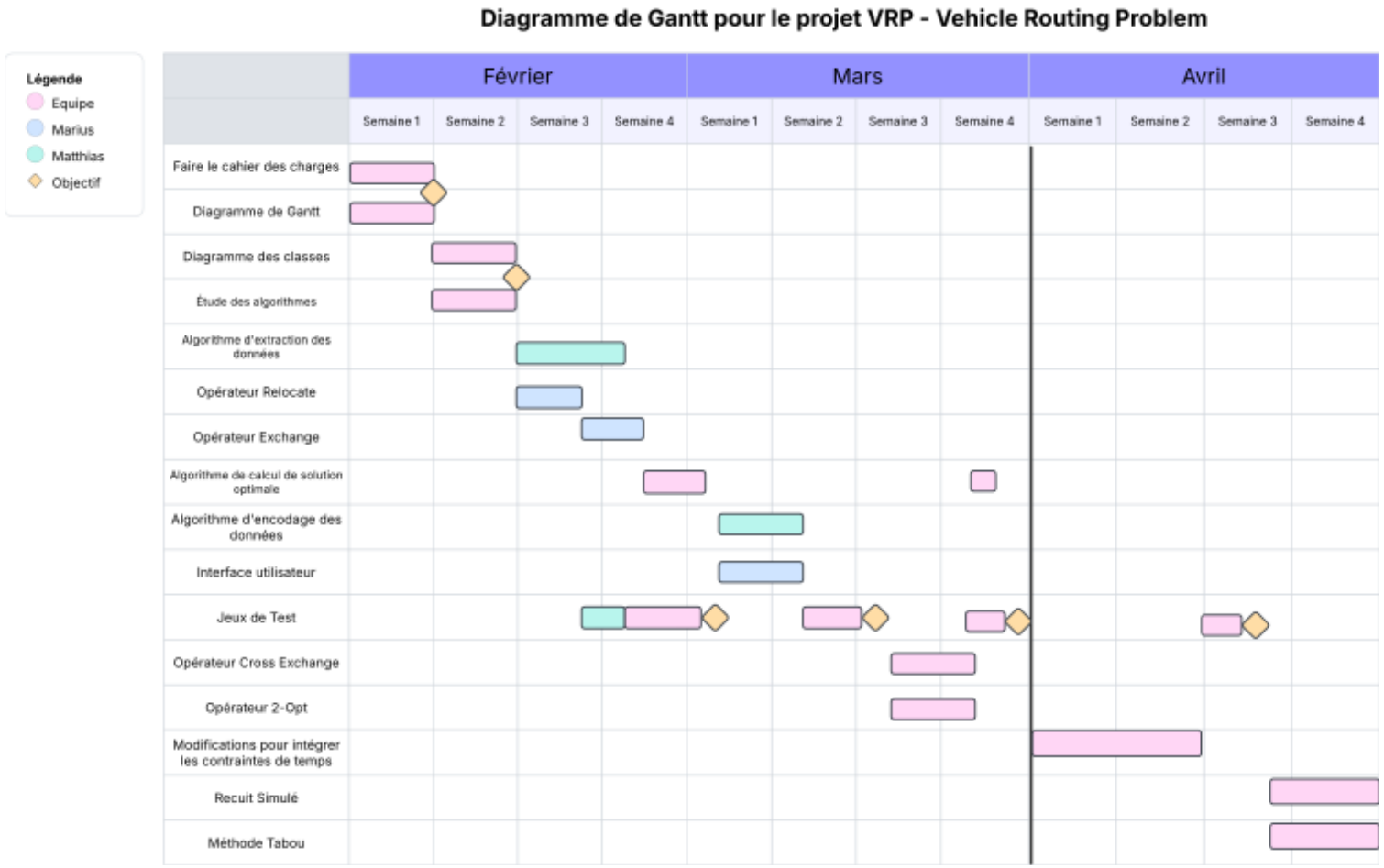
Le projet présenté a permis de développer une application efficace pour résoudre le Capacitated Vehicle Routing Problem with Time Windows (CVRPTW), un problème d'optimisation logistique crucial dans le domaine de la distribution. En s'appuyant sur un algorithme génétique, nous avons réussi à générer des tournées de livraison optimisées, respectant les contraintes de capacité des véhicules et les fenêtres horaires imposées par les clients. Les résultats obtenus montrent une réduction significative de la distance parcourue, atteignant jusqu'à 84,96 % dans des scénarios sans contraintes, et environ 67,47 % lorsque toutes les contraintes sont appliquées.
La méthodologie adoptée a permis de structurer le projet de manière à garantir une répartition efficace des tâches et une communication fluide au sein de l'équipe. Cette approche a été essentielle pour respecter les délais et assurer la qualité du produit final.

Pistes d’amélioration

Pour l'avenir, plusieurs pistes d'amélioration peuvent être envisagées. La transformation de l’algorithme actuel vers la méthode Tabou ou le Recuit Simulé permettrai d’atteindre de meilleurs gains, grâce à leur faculté de ne pas tomber dans le premier minimum local encontré. De plus, l'intégration de contraintes supplémentaires, telles que : remplacer le plan que traversent les véhicules par un graphe, accroître le nombre de dépôts, ou encore contraindre les marchandises rentrées en premier à ressortir en dernier, permettrait d'accroître la pertinence et l'applicabilité de l'outil dans des contextes réels. Enfin, une exploration plus approfondie des dynamiques sous-jacentes, comme la relation de proportionnalité entre le nombre de clients et le nombre d’itérations de l’algorithme, pourrait offrir des perspectives intéressantes.

Annexes

Annexe 1 : Diagramme de Gantt du projet



Annexe 2 : Illustrations des opérateurs