Projet Mathématiques - Informatique

Sami SOGHAIER
Gabriel VALDES-ALONZO

Licence 3 - Mathématiques-Informatique
Maria Raquel URENA PEREZ

Juin 2025

Introduction

Le problème du voyageur de commerce [1, 2] (Traveling Salesman Problem en anglais) est un problème classique d'optimisation combinatoire : le problème consiste à trouver le plus court chemin passant une seule fois par une liste de n villes définies, et revenant au point initial.

Le TSP a diverses applications dans plusieurs domaines comme la logistique, séquençage d'ADN, manufacture, et autres. Chaque problème composé d'un ensemble qui doit être rempli dans un ordre particulier pourrait être pensé comme le problème TSP, où on cherche un chemin moins coûteux. Le problème peut être résolu exactement si tous les chemins entre villes sont analysés et comparés, mais la quantité de chemins à vérifier augmente rapidement, ce qui est connu comme un problème NP-Difficile.

Dû à son coût, des algorithmes sont proposés pour trouver des solutions approximatives qui peuvent être optimales ou non-optimales, mais qui résoud le problème rapidement. Dans ce travail, seulement la version symétrique de l'algorithme sera analysé. Cela veut dire que les distances considérées entre villes sont égales dans les deux sens. Les villes sont aussi considérées toutes interconnectées, donc il existera toujours une connexion directe entre une ville et l'autre.

Le travail a été réparti de la façon suivante : chaque personne a pris deux algorithmes à implémenter. L'heuristique du plus proche voisin ainsi que 2-opt sont une continuité, comme l'heuristique d'Insertion qui fonctionne avec le recuit simulé. En ce qui concerne le rapport, il a été écrit en développant chacun ses parties respectives et en faisant les analyses ensemble.

Dans les prochaines sections, on procédera ensuite à l'analyse des résultats et puis on finira par conclure en comparant les résultats et en regardant les perspectives futures.

Divers outils ont été utilisés pour travailler sur le rapport :

— Overleaf : pour rédiger le rapport et travailler ensemble en temps réel,
— Notebook : pour rédiger le notebook jupyter,
— VSCode : pour travailler sur le code du projet,
— Github : pour stocker et échanger les différents supports.

Le dépôt Github peut être consulté sur le lien suivant :
`https://github.com/grvaldes/EADS-projet-mathematique`.

Heuristiques implémentées

Dans le contexte de ce travail sur le TSP, on va mettre en pratique des heuristiques afin d'apporter des solutions qui soient les plus correctes possibles en minimisant le temps de calcul. Les heuristiques analysées sont les suivantes :

Voisins Plus Proches

L'heuristique VPP [3], aussi connu par son nom en anglais Nearest Neighbors, est un des algorithmes les plus simples et intuitifs pour résoudre le TSP. Comme son nom le dit, étant donnée une ville de départ quelconque (choisi arbitrairement), on cherche entre toutes les villes

voisines laquelle est la plus proche. Une fois dans cette ville, on la marque comme visitée et on continue en cherchant la ville la plus proche encore non visitée. On répète cette procédure jusqu'à ce qu'on ait visité toutes les villes dans le réseau. La complexité de cet algorithme est exactement $\mathcal{O}(n^2)$ (avec n la quantité de villes), car pour chaque ville il faut vérifier la distance avec chaque voisin.

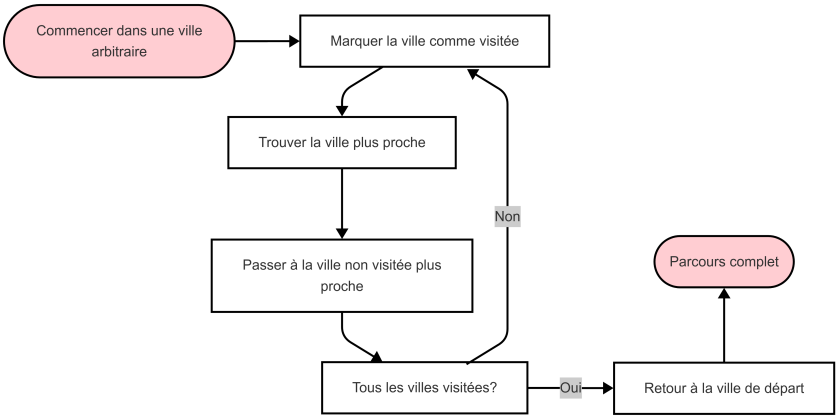

FIGURE 1 – Diagramme de l'algorithme.

Heuristique d'Insertion

L'heuristique d'insertion [4] est une méthode qui consiste à construire une solution de manière progressive en insérant les villes les unes après les autres dans un réseau selon une règle de sélection qui minimise le coût. C'est une solution simple et rapide qui est très efficace sur des réseaux qui ont des tailles moyennes. L'heuristique d'insertion sert également de base pour le recuit simulé.

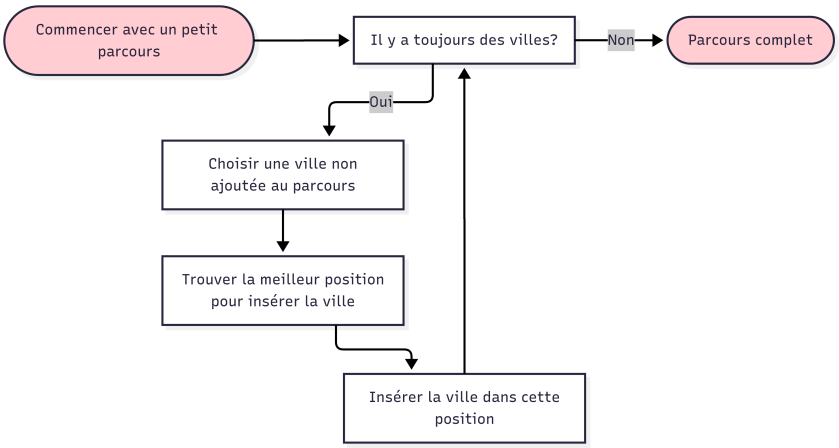

FIGURE 2 – Diagramme de l'algorithme.

2-opt

L'heuristique 2-opt [5] est aussi une procédure simplifiée pour chercher le chemin le plus optimal. Pour un chemin donné, l'algorithme cherche à améliorer le parcours en inversant des sous-chemins. Donc pour chaque ville, l'algorithme cherche le parcours à suivre et vérifie si la distance totale est réduite quand on inverse une partie parcourue. Si l'on trouve un parcours plus court, on garde ce nouveau chemin et on commence une fois encore à partir de la ville suivante, et on répète jusqu'à avoir trouvé le chemin optimal. La quantité de sous-chemins à vérifier est sous-quadratique, mais chaque sous-chemin a une quantité différente de villes, donc la complexité de l'algorithme peut exploser facilement.

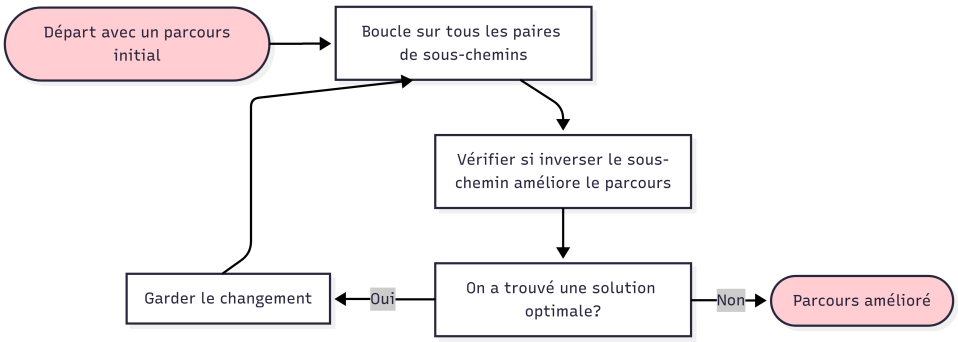

FIGURE 3 – Diagramme de l'algorithme.

Recuit Simulé

Le recuit simulé [6]provient du procédé de fabrication des métaux où le refroidissement plus ou moins lent favorise le durcissement et la stabilité du métal. Cette heuristique fonctionne sur le même principe. Le recuit simulé consiste à utiliser comme point de départ une solution déjà proposée (ici, l'heuristique d'insertion) et à l'améliorer en utilisant des solutions temporaires même si elles ne sont pas optimales afin d'éviter les minima-locaux.

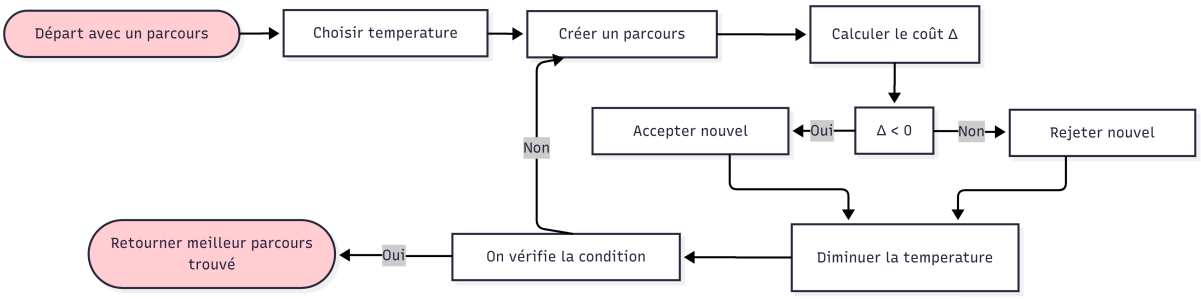

FIGURE 4 – Diagramme de l'algorithme.

Résultats

On présente les résultats obtenus de l'application des algorithmes dans des réseaux de villes. Le code est organisé de cette façon : on peut donner une graine aléatoire qui génère toujours le même réseau, ce qui facilite les comparaisons entre les différents algorithmes.

Voisins plus proches

Pour essayer l'algorithme des voisins plus proches, on génère un réseau de 50 villes avec une graine égale à 30. On utilise l'algorithme avec trois points de départ différents : la ville 0, 23 et 42 :

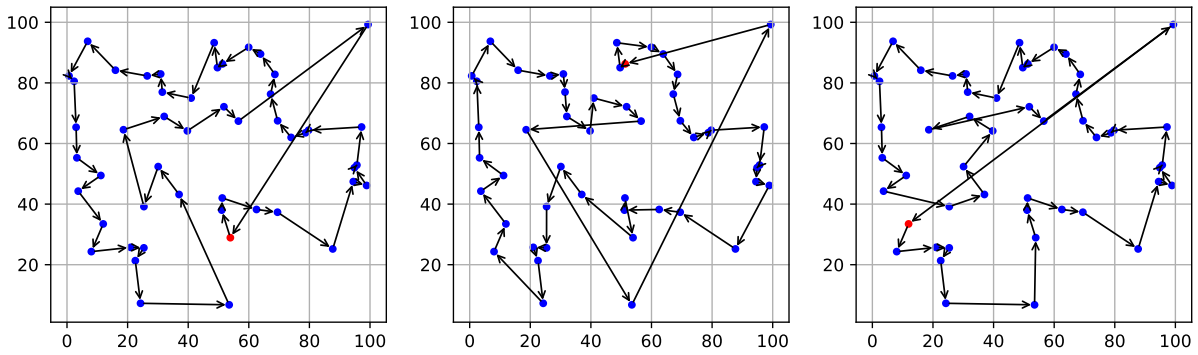

FIGURE 5 – Résultat pour l'algorithme des Voisins plus proches avec trois départs différents (50 villes). Les départs depuis la ville 0 (gauche), 23 (centre) et 42 (droite).

On a que l'algorithme prend en moyenne 299.61 microsecondes, et pour chaque point de départ on parcourt :

— Départ ville 0 : 669.16.
— Départ ville 23 : 715.63.
— Départ ville 43 : 692.43.

Après ces résultats, on peut vérifier que l'optimalité de l'algorithme dépend du point de départ, donc il faudrait appliquer l'heuristique 50 fois pour obtenir la meilleure solution possible avec cet algorithme. En particulier, on a une ville qui est trop loin des autres, et cela cause de problèmes avec l'optimalité du chemin parcouru (elle génère des croisements dans le chemin).

2-opt

Pour l'algorithme 2-opt on utilise le même réseau que dans l'heuristique précédent, pour comparer les résultats, avec un départ de la ville 0. Si pour les voisins proches on avait une distance totale de 669.16, avec l'algorithme 2-opt on réduit cette distance à 610.09. Si l'on regarde la Figure 6 on voit que cela est dû à l'élimination des croisements, en obtenant un parcours plus direct. Cependant, on a l'algorithme 2-opt qui est plus lent. Dans cet exemple, on obtient un temps d'exécution de 71.25 millisecondes, qui est trois ordres de magnitude plus grand.

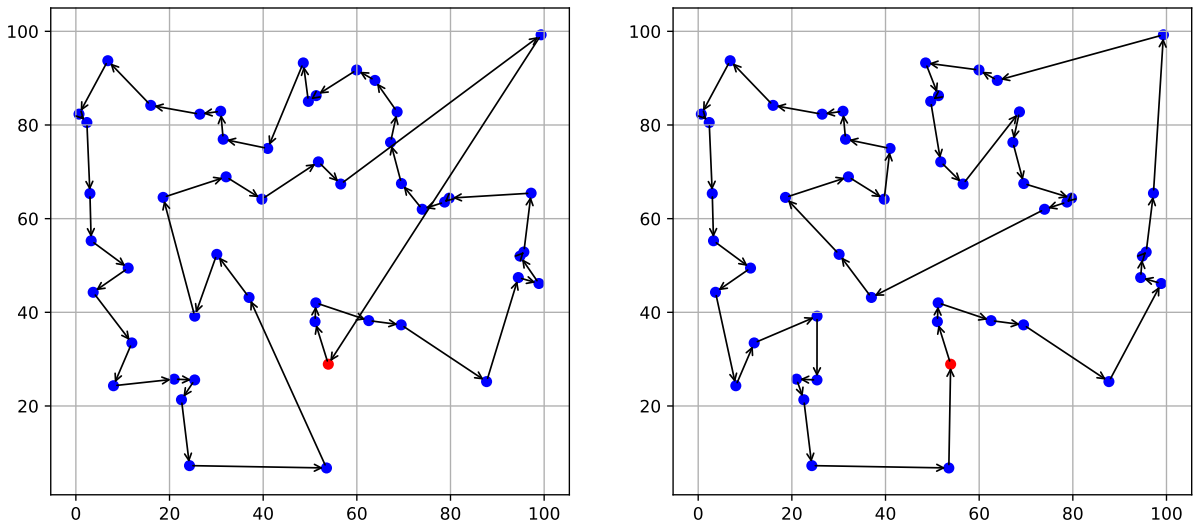

FIGURE 6 – Résultat pour l'algorithme 2-opt (gauche) et sa comparaison avec le résultat des Voisins plus proches (droite) pour 50 villes.

Si l'on applique l'heuristique 2-opt avec une autre solution de voisins plus proches (dans ce cas le même réseau avec départ de la ville 23) on a la distance qui diminue de 715.63 à 607.72 mais le parcours est différent, ce qui dit que l'optimalité de l'algorithme 2-opt dépend également de son initialisation.

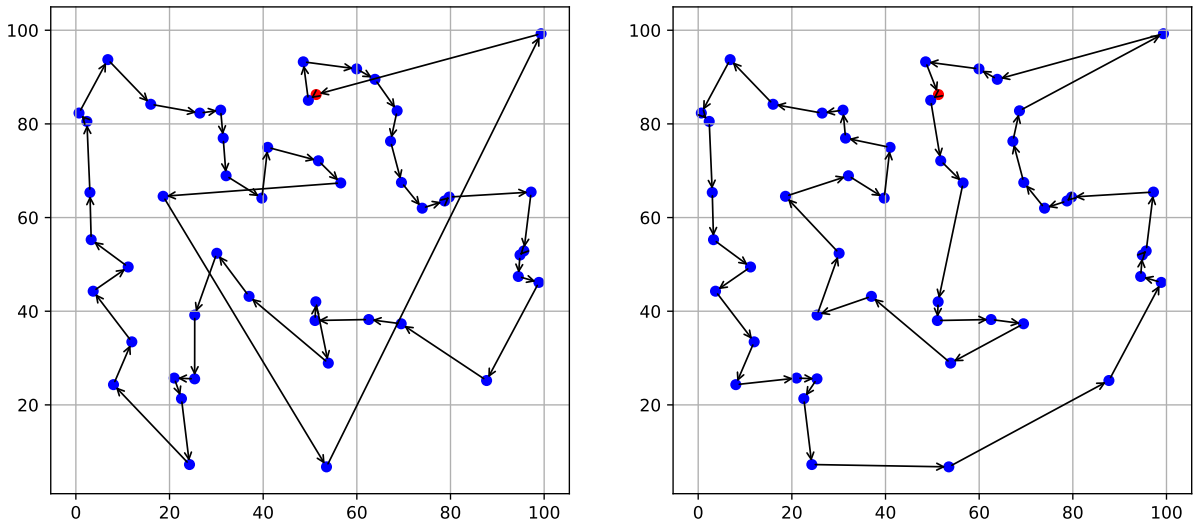

FIGURE 7 – Résultat de 2-opt et sa comparaison avec le résultat des Voisins plus proches pour un départ au milieu.

Finalement, si l'on prend le parcours avec le début dans la ville 42, on a une diminution de la distance parcourue de 692.44 à 609.91, avec l'élimination du grand croisement entre les deux dernières villes.

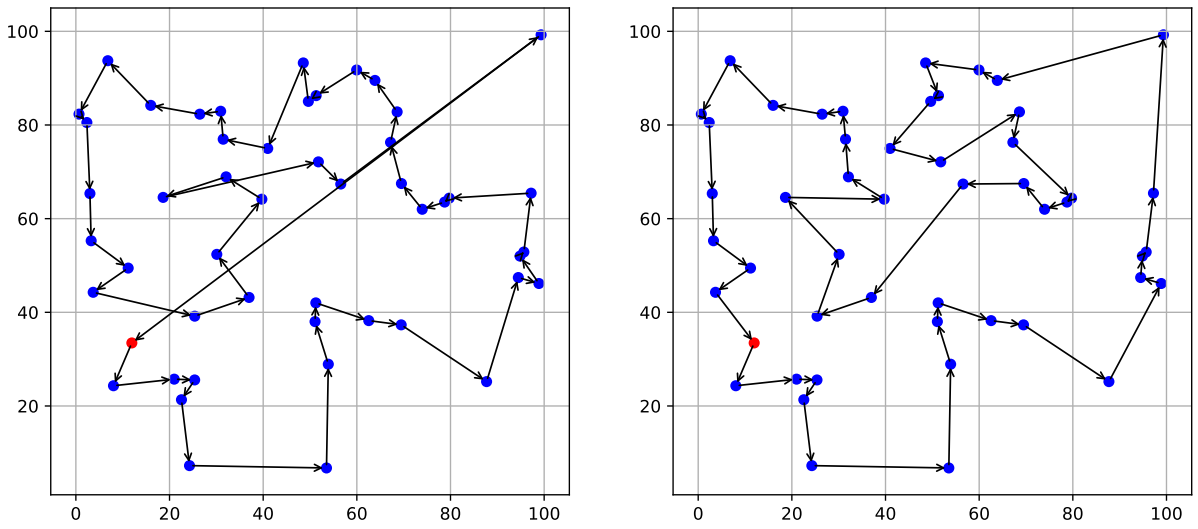

FIGURE 8 – Résultat de 2-opt pour un départ différent. On vérifie que le résultat de 2-opt est dépendant du résultat initial des voisins plus proches, donc le résultat n'est pas fixé.

Heuristique d'insertion

Pour l'heuristique d'insertion avec 20 villes, on obtient un graphe fluide avec une distance totale de 355.04 pour un temps de calcul de 535 microsecondes, que l'on voit sur la Figure 9

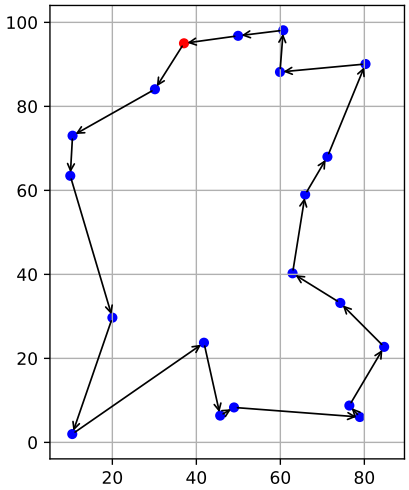

FIGURE 9 – Résultat pour l'heuristique d'insertion avec 20 villes.

Les résultats obtenus pour l'heuristique d'insertion avec 100 villes sont les suivants :

— distance totale : 930.16,
— temps de calcul : 24.143 millisecondes,

tandis que les résultats pour l'heuristique de voisins plus proches sont :

— distance totale : 1062.79,
— temps de calcul : 0.609 millisecondes.

L'algorithme VPP est certe plus rapide que le premier mais il prendre aussi un chemin un peu plus long, et comme on peut le voir sur la figure ci-après, l'heuristique d'insertion ne coupe pas le chemin contrairement au VPP ce qui le rend plus agréable à étudier.

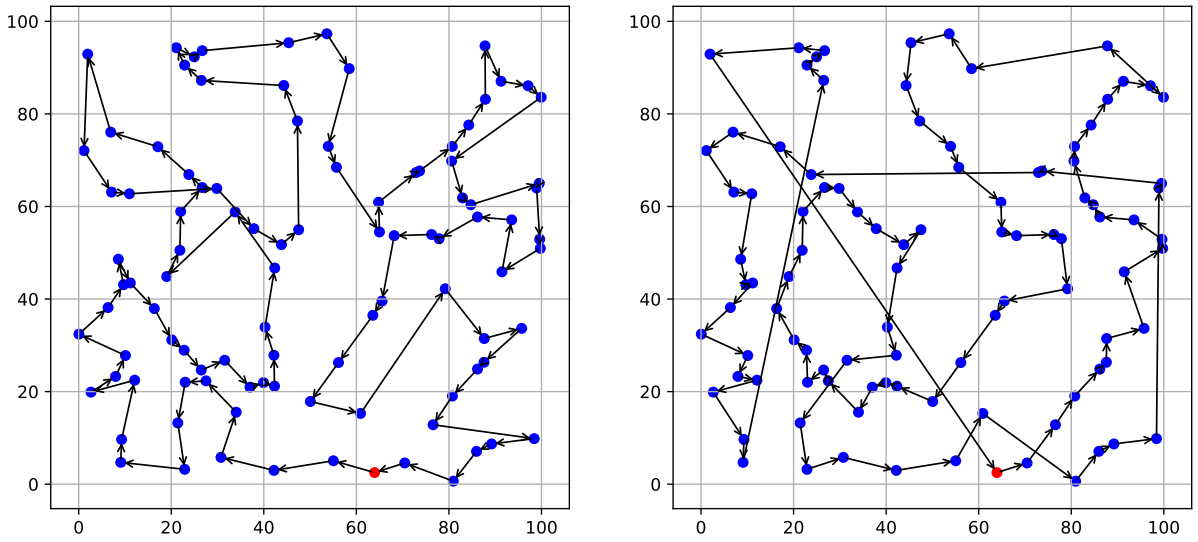

FIGURE 10 – Résultat pour l'heuristique d'insertion avec 100 villes et une comparaison avec le résultat des Voisins plus proches pour le même réseau.

Recuit simulé

Les mesures suivantes ont été obtenus pour l'heuristique d'insertion :

— distance totale : 897.78,
— temps de calcul : 31.324 millisecondes,

contre ces résultats pour le recuit simulé :

— distance totale : 865.26,
— temps de calcul : 308.325 millisecondes.

Comme on le constate sur la Figure 11, même si le temps de calcul augmente et est multiplié par 10 par rapport à l'heuristique d'insertion, la distance du chemin est diminuée et optimisée. Le temps de calcul reste très faible pour cette quantité de villes : environ 300 millisecondes, ce qui rend le recuit simulé optimal pour des calculs de chemins avec un nombre de villes important.

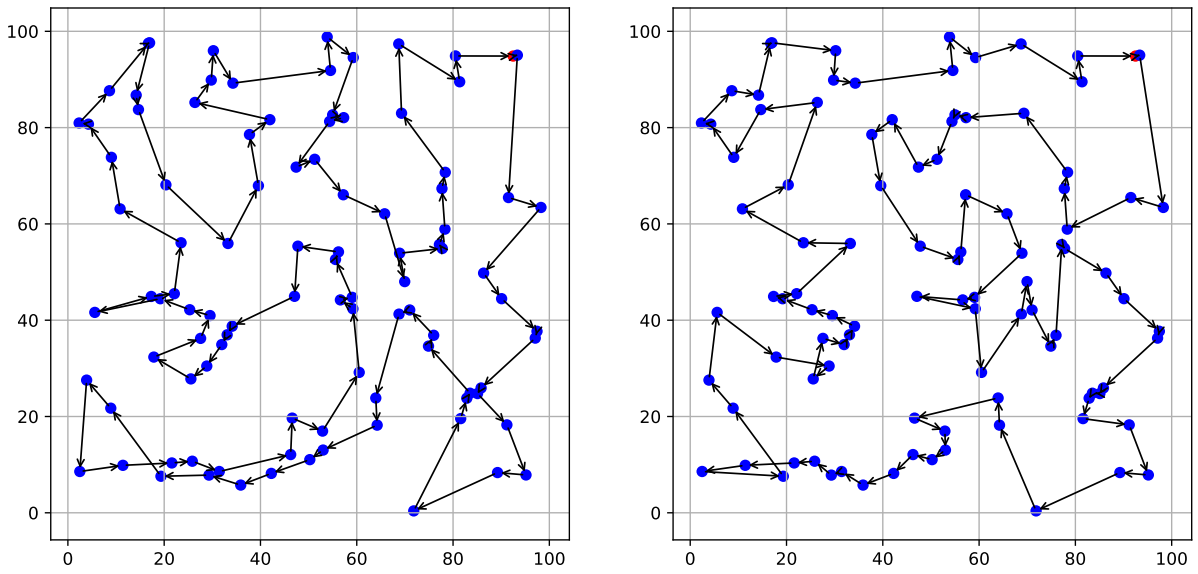

FIGURE 11 – Résultat pour l'heuristique d'insertion (à gauche) et son amélioration par l'algorithme de Recuit simulé (à droite) avec 100 villes.

Comparaison entre algorithmes

Les différents algorithmes analysés ont leurs avantages et leurs problèmes, mais sauf cas particuliers, on ne les compare pas directement. Ici, on prend des réseaux avec une graine choisie aléatoirement (3456) de différentes dimensions : on génère des réseaux de 10, 50, 100, 500 et 1000 villes et l'on applique les différentes heuristiques pour trouver les meilleures solutions de chaque algorithme. Les résultats sont montrés dans le tableau 1 :

TABLE 1 – Distance obtenue avec les différents algorithmes pour une quantité N de villes.

N villes	VPP	2-opt	Heuristique d'insertion	Recuit simulé
10	286.2587	266.1043	299.9325	266.1043
50	588.5159	549.2188	615.9452	565.7999
100	888.5622	-	853.6044	827.6085
500	2099.8537	-	1936.8251	1936.8251
1000	2916.9891	-	2716.844	2716.8443

La première chose à noter est le manque de données pour l'algorithme 2-opt. Cela est dû à l'inefficience de l'algorithme. Pour un réseau de 100 villes l'algorithme ne converge pas après 5 minutes, donc il est inutile d'attendre car l'incrémentation de temps pour les réseaux plus grands rendra impossible la solution. Cependant, on peut voir que dans les cases où l'heuristique 2-opt est présente, il donne la solution la plus optimale au problème pour ce réseau particulier. C'est intéressant de noter aussi que pour les réseaux plus petits l'heuristique d'insertion n'est pas plus optimale que les voisins plus proches, mais avec la croissance des réseaux on a une amélioration de la performance de l'heuristique d'insertion. Mais comme on a déjà vu, l'algorithme des Voisins plus proches est plus rapide, il donc est intéressant aussi de faire un compromis entre temps et précision.

Complexité temporelle

Finalement, on fait une analyse temporelle des algorithmes pour voir la tendance en fonction de la taille des réseaux. Pour ce cas, on utilise les algorithmes pour résoudre le TSP pour des réseaux de 10, 50, 100, 500 et 1000 villes, mais cette fois-ci on génère des réseaux aléatoires et pour chaque taille on résoud le problème 5 fois, pour avoir une signifiance statistique. Les résultats sont visibles dans la Figure 12 :

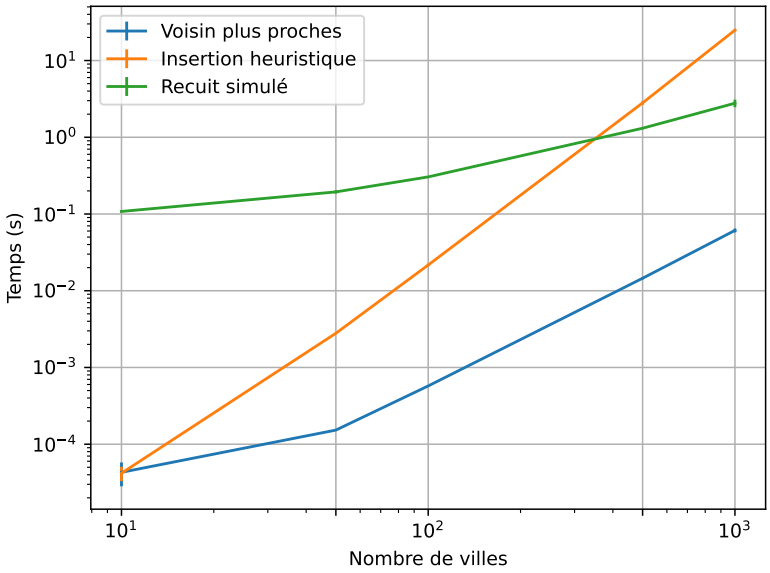

FIGURE 12 – Nombre de villes vs. le temps moyenne d'exécution des algorithmes.

On montre les courbes dans un espace logarithmique pour faire une régression polynomiale qui permette d'obtenir une droite pour chaque courbe. Ces droites ont pour pente :

— Voisins plus proches : 1.636,

— Heuristique d'insertion : 2.902,

— Recuit simulé : 0.721.

On peut voir que l'algorithme de voisins plus proches, pour les cas que l'on a analysé a une complexité de $\mathcal{O}(n^{1.636})$, qui est dans l'ordre de magnitude de n^2, le pire cas. Pour l'heuristique d'insertion, on obtient $\mathcal{O}(n^{2.902})$ qui est pire que le n^2 attendu, mais dans une marge acceptable. Finalement, on voit que le Recuit simulé donne une complexité $\mathcal{O}(n^{0.721})$, qui est très efficient en comparaison des autres algorithmes. On peut attendre que pour des réseaux très grands, ce serait l'option la plus avantageuse, mais il faut considérer quand même que pour utiliser cette stratégie, on doit utiliser une autre heuristique avant pour démarrer l'analyse.

Conclusion

Dans ce project, le problème du voyageur de commerce a été analysé et quelques heuristiques de résolution ont été présentés. Divers outils informatique ont été utilisés pour le développement de ce projet, tels comme Overleaf pour écrire le rapport et pouvoir y travailler et y apporter des modifications en temps réel, l'utilisation de Python et Notebook pour rendre un notebook Jupyter et GitHub afin de collaborer de manière efficace. Chacun à travaillé sur ces heuristiques respectives puis les résultats ont été mis en commun afin de les comprendre et de les comparer.

Il ressort de cette étude des heuristiques que chacune peut être utile selon le degré d'acceptation mis en place, c'est-à-dire si l'on souhaite un résultat rapide, un résultat court ou bien un résultat équilibré. Le recuit simulé est dans l'ensemble la meilleure heuristique puisqu'en cas de gros réseaux, on a un résultat équilibré : il est assez long en temps de calcul mais la distance est aussi moindre. L'heuristique 2-opt est aussi efficace puisqu'elle améliore l'heuristique VPP mais tout cela dépend de son initialisation. Par contre, en cas de petits réseaux, l'heuristique VPP est plus efficace que l'heuristique d'insertion, particulièrement si le temps d'exécution est une facteur importante a considérer.

Les problèmes rencontrés sur le projet ont été le travail à distance pour bien s'accorder sur la façon de travailler et d'avoir les mêmes points de départ afin de comparer les heuristiques. Un autre problème a été le temps de traitement et de calcul de l'heuristique 2-opt qui était très long. Dans un future implementation, il est nécessaire de vérifier avec soin le code et de l'optimiser. Python est utile pour une première tentative, mais pour une solution plus efficace il y a des autres langages mieux adaptés pour ces tâches. Même dans Python, il a des possibilités pour améliorer la solution : tous le travail a été sans l'utilisation de librairies optimisés tel comme numpy, qui permet la manipulation efficient de matrices.

Pour conclure, ce projet a été très intéressant à traiter, il permette de travailler en équipe et d'approfondir nos connaissances sur les algorithmes, et en particulier de découvrir les heuristiques liées au problème du voyageur de commerce, et de faire les premiers pas dans le domain de l'optimisation.

Références

[1] Donald Davendra. *Traveling Salesman Problem, Theory and Applications.* InTech, Janeza Trdine 9, 51000 Rijeka, Croatia, first edition, 2010.

[2] Karla Hoffman et al. *Traveling Salesman Problem (TSP)Traveling salesman problem*, pages 849–853. Springer US, New York, NY, 2001.

[3] Cor A.J. Hurkens and Gerhard J. Woeginger. On the nearest neighbor rule for the traveling salesman problem. *Operations Research Letters*, 32(1) :1–4, 2004.

[4] Gilbert Laporte Clazude Gendreau, Alain Hertz. New insertion and post-optimization procedures for the traveling salesman problem. 1992.

[5] Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. *Algorithmica*, 68(1) :190–264, 2014.

[6] Jean-Yves Potvin Michel Gendreau. Handbook of metaheuristics, 2nd éd. 2010.