CESI
ÉCOLE D'INGÉNIEURS

A3 FISA INFORMATIQUE

Recherche Opérationnel

PARTIE 2 : MODELISATION

Charly OLIVIER
Youssef AMALLAH
Yaël WOJCIK
Guillaume HOUNKPATI

Table des matières

1. Glossaire

- **Problème du Voyageur de Commerce (TSP -** Traveling Salesman Problem): Problème algorithmique consistant à trouver le chemin le plus court pour visiter un ensemble de villes et revenir au point de départ, en minimisant la distance totale parcourue.

- **Problème de Tournée des Véhicules (VRP - Vehicle Routing Problem):** Extension du TSP visant à optimiser les itinéraires d'une flotte de véhicules pour livrer des marchandises à divers clients tout en minimisant les coûts.

- **Graphes:** Structures mathématiques utilisées pour modéliser des réseaux de routes dans le cadre du TSP et du VRP, où les sommets représentent des villes et les arêtes des routes.

- **Heuristiques:** Méthodes approximatives utilisées pour trouver des solutions satisfaisantes à des problèmes complexes comme le TSP et le VRP en un temps raisonnable.

- **Méthodes Exactes:** Algorithmes qui garantissent de trouver la solution optimale d'un problème, souvent au prix d'un temps de calcul plus élevé, par exemple, la programmation linéaire.

- **2-opt:** Technique d'amélioration des solutions heuristiques par l'échange de deux arêtes dans le but de réduire la distance totale parcourue.

- **Recherche Tabou:** Méthode d'optimisation qui explore l'espace des solutions en tenant compte des solutions précédemment visitées pour éviter les cycles et améliorer les résultats.

- **Complexité NP-difficile:** Catégorie de problèmes pour lesquels aucune solution efficace (en temps polynomial) n'est connue pour toutes les instances, nécessitant souvent des heuristiques pour des solutions pratiques.

- **Programmation Linéaire:** Technique mathématique pour optimiser une fonction objective sous des contraintes linéaires, utilisée pour modéliser et résoudre des problèmes comme le TSP et le VRP.

- **PuLP, CPLEX, Gurobi:** Solveurs et bibliothèques en Python utilisés pour résoudre des modèles de programmation linéaire et mixte-entière.

2. Introduction

2.1 Présentation générale du contexte du projet

L'Agence de l'Environnement et de la Maîtrise de l'Énergie (ADEME) a lancé un appel à manifestation d'intérêt pour encourager la réalisation de démonstrateurs et d'expérimentations de nouvelles solutions de mobilité pour les personnes et les marchandises adaptées à divers types de territoires. Cette initiative s'inscrit dans un contexte où la transition vers des solutions de transport plus durables et intelligentes est essentielle pour réduire les émissions de gaz à effet de serre, améliorer la qualité de l'air et optimiser l'utilisation des ressources énergétiques.

La structure CesiCDP, déjà bien ancrée dans le domaine de la Mobilité Multimodale Intelligente, s'associe à cette initiative pour proposer des solutions innovantes. Avec l'aide de nombreux partenaires, CesiCDP a mené plusieurs études sur les nouvelles technologies de transport, mettant en évidence des solutions plus économiques et moins polluantes. Cependant, ces nouvelles technologies présentent également des défis, notamment en matière d'optimisation de la gestion des ressources et de logistique.

Les défis logistiques du transport sont cruciaux pour l'avenir, car ils ont de multiples applications : distribution du courrier, livraison de produits, gestion du réseau routier, et ramassage des ordures. L'impact environnemental de ces activités peut être considérable, d'où l'importance de développer des solutions de transport plus efficaces et moins polluantes.

L'appel de l'ADEME représente une opportunité stratégique pour CesiCDP de sécuriser de nouveaux marchés et des financements substantiels, permettant de continuer à développer et à déployer des solutions de mobilité innovantes. La participation à ce projet s'aligne parfaitement avec les objectifs de développement durable et d'innovation technologique de CesiCDP.

2.2 Présentation générale du problématique du projet

L'objectif principal de ce projet est de minimiser les déplacements et la consommation des véhicules lors des livraisons, ce qui nécessite l'optimisation des tournées de transport sur un réseau routier. La problématique se pose sous la forme d'un problème algorithmique classique connu sous le nom de Problème du Voyageur de Commerce (TSP - Traveling Salesman Problem). Le défi consiste à calculer une tournée permettant de relier un sous-ensemble de villes tout en revenant au point de départ, de manière à minimiser la durée totale de la tournée.

Ce problème, bien que simple à énoncer, est réputé pour sa complexité algorithmique. Il s'agit d'un problème NP-difficile, ce qui signifie qu'il n'existe pas de solution efficace (en temps polynomial) pour toutes les instances possibles du problème. Cela nécessite donc l'utilisation de méthodes heuristiques ou d'algorithmes d'approximation pour trouver des solutions satisfaisantes dans des délais raisonnables, surtout pour des instances de taille réaliste.

Les défis spécifiques du projet incluent :

- Modélisation du problème : Représenter les données du réseau routier, les distances entre les villes et les contraintes spécifiques des tournées de livraison.
- Analyse de la complexité : Comprendre et caractériser la complexité du problème pour adapter les approches algorithmiques en conséquence.
- Développement du code en Python : Créer un algorithme capable de résoudre des instances réalistes du problème. Cet algorithme doit être performant et capable de traiter des données réelles avec efficacité.
- Étude statistique du comportement expérimental : Analyser les performances de l'algorithme développé sur différentes instances, afin de mesurer son efficacité, sa robustesse et les conditions dans lesquelles il offre les meilleures performances.

La solution proposée par CesiCDP devra non seulement être techniquement solide, mais aussi adaptable à diverses situations de transport et types de territoires, afin de maximiser son impact environnemental et économique.

2.3 Contraintes

Les contraintes dans le problème du voyageur de commerce (TSP) sont essentielles pour définir et résoudre le problème correctement. Les contraintes principales incluent :

- Chaque ville doit être visitée exactement une fois, ce qui empêche les cycles ou sous-tours inutiles qui ne couvrent pas toutes les villes.
- Une contrainte importante est d'éliminer les trajets d'une ville à elle-même, garantissant ainsi qu'il n'y a pas de boucle.
- Une autre contrainte stipule que le voyageur doit commencer et terminer à un point de départ fixe, souvent appelé le dépôt.

3. Modélisation

3.1 Théorie des graphes

Pour résoudre efficacement le problème d'optimisation des tournées de livraison, nous nous appuyons sur le concept mathématique des graphes. Cela nous permet de représenter de manière abstraite et structurée la configuration d'une carte routière. Une fois cette représentation formalisée, nous pourrons créer une structure de données adaptée pour alimenter notre algorithme.

- **Modélisation**

Dans la phase de modélisation, nous allons nous concentrer sur la conversion de la carte en un graphe. Dans ce contexte, chaque point de livraison sera représenté par un sommet du graphe. Les arêtes, représentant les routes entre les points, seront générées de manière aléatoire. L'objectif est de visiter tous les points de livraison tout en minimisant la distance totale parcourue.

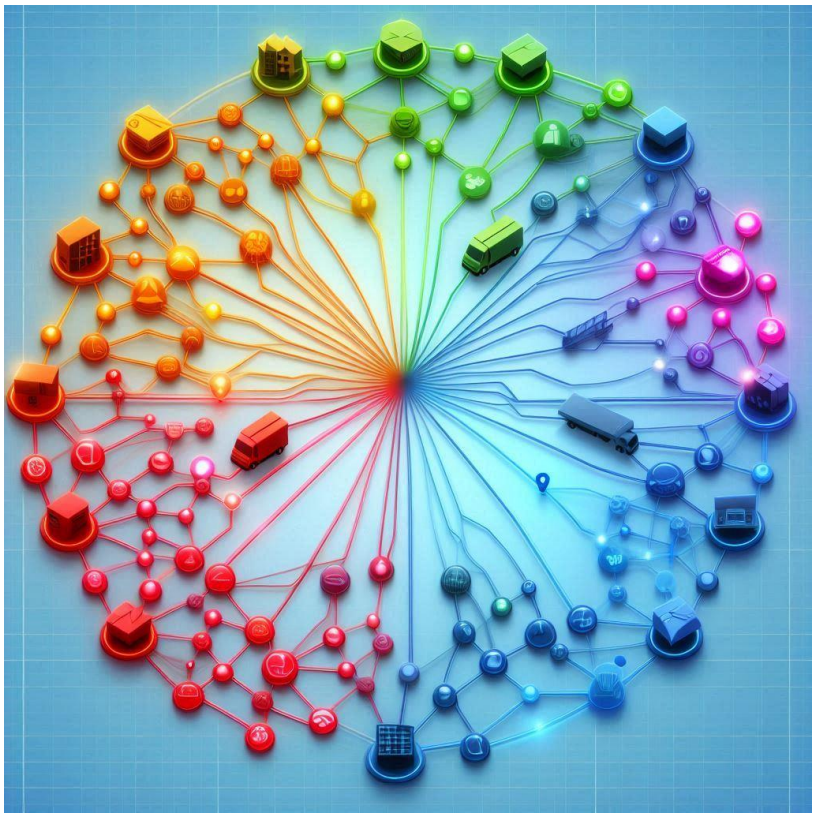


Figure 1 : Modélisation

- **Choix de l'Algorithme**

Après avoir établi la modélisation, nous déterminerons quel type d'algorithme est le plus approprié pour résoudre ce problème de manière efficace. L'objectif est de gérer un grand nombre de points de livraison en maintenant une complexité algorithmique raisonnable.

3.2 Modélisation Linéaire du Problème

Le problème consiste à calculer une tournée optimale sur un réseau routier, connectant un ensemble de villes et retournant au point de départ, afin de minimiser la durée totale de la tournée. Voici les principaux éléments de la modélisation :

Variables de décision:

1. X_{ij} : Variable binaire qui est égale à 1 si le trajet va de la ville i à la ville j, sinon 0.
2. U_i : Variable continue utilisée pour éliminer les sous-tours (sub-tour elimination).

Fonction Objective:

Minimiser la distance totale parcourue:

$\text{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} X_{ij}$

où d_{ij} est la distance entre les villes i et j.

Contraintes:

1. Chaque ville doit être visitée exactement une fois (devenir un point de départ):
   $\sum_{j=1, j \neq i}^{n} X_{ij} = 1 \quad \forall i$
2. Chaque ville doit être visitée exactement une fois (devenir un point d'arrivée):
   $\sum_{i=1, i \neq j}^{n} X_{ij} = 1 \quad \forall j$
3. Élimination des sous-tours (Sub-tour elimination) pour éviter que le tour ne se divise en sous-tours plus petits:
   $U_i - U_j + nX_{ij} \leq n - 1 \quad \forall i, j \ (i \neq j, i \neq 0, j \neq 0)$
4. Contrainte sur U_i:
   $1 \leq U_i \leq n - 1 \quad \forall i \ (i \neq 0)$
5. Contraintes de binaire sur X_{ij}:
   $X_{ij} \in \{0, 1\}$

Pour résoudre ce problème de tournée de véhicules, une approche utilisant la programmation linéaire ou mixte-entière peut être adoptée. Voici les étapes typiques :

- **Formulation :** Traduire les contraintes et la fonction objectif en un modèle de programmation linéaire.
- **Solutions Techniques :** Utiliser des solveurs tels que CPLEX, Gurobi, ou des bibliothèques comme PuLP en Python pour résoudre le modèle.
- **Optimisation :** Explorer différentes techniques de résolution pour améliorer les performances et l'efficacité de la solution.

3.3 Complexité

Le problème du Voyageur de Commerce (Traveling Salesman Problem, TSP) est l'un des problèmes les plus étudiés en recherche opérationnelle et en informatique. Il s'agit de trouver le chemin le plus court possible qui visite chaque ville exactement une fois et revient à la ville de départ. La complexité de ce problème est due à plusieurs facteurs, que nous allons explorer en détail :

1. Solution Exacte

La solution exacte au TSP peut être trouvée en utilisant des méthodes de programmation linéaire et des solveurs spécifiques. Cependant, en raison de la nature NP-difficile du problème, ces méthodes ne sont efficaces que pour de petites instances. La complexité de la solution exacte augmente de façon exponentielle avec le nombre de nœuds, ce qui signifie que le temps de calcul devient impraticable pour des instances de taille moyenne à grande.

a. Modèle Mathématique

Le modèle mathématique du TSP utilise des variables binaires xij pour indiquer si le chemin va du nœud i au nœud j. Les contraintes garantissent que chaque nœud est visité exactement une fois, que le camion quitte et revient au dépôt, et évitent les sous-tours. Les contraintes temporelles assurent que le trajet respecte l'ordre des visites.

b. Solveurs Utilisés

Les solveurs comme PuLP, CPLEX, et GUROBI sont des outils puissants pour résoudre des modèles de programmation linéaire. Cependant, ils peuvent rapidement devenir inefficaces lorsque le nombre de nœuds dépasse une certaine limite (généralement autour de 15 à 20 nœuds), en raison de la combinatoire exponentielle du problème (complexité en O(n!)).

2. Heuristiques et Métaheuristiques

Pour des instances plus grandes, où les solutions exactes sont impraticables, on utilise des heuristiques et des métaheuristiques. Ces méthodes n'offrent pas une garantie de trouver la solution optimale, mais elles peuvent produire des solutions suffisamment bonnes en un temps raisonnable.

a. Heuristiques de Construction

4. **Heuristique du Plus Proche Voisin** : Sélectionne toujours le nœud le plus proche non visité. Simple à implémenter, mais peut mener à des solutions sous-optimales.
5. **Heuristique d'Insertion la Plus Proche** : Insère le nœud le plus proche au meilleur endroit possible dans le trajet.

6. **Heuristique d'Insertion la Moins Coûteuse** : Examine tous les nœuds pour trouver l'insertion la moins coûteuse.
7. **Heuristique d'Insertion la Plus Éloignée** : Insère le nœud le plus éloigné, ce qui peut aider à éviter les mauvais départs locaux.
8. **Heuristique de MST** : Utilise un arbre couvrant minimal pour construire une solution initiale.

b. Heuristiques d'Amélioration

9. **Échange 2-Opt** : Améliore une solution existante en échangeant deux arêtes pour réduire le coût total du trajet. C'est une méthode efficace pour améliorer les solutions trouvées par des heuristiques de construction.

c. Métaheuristiques

10. **Recherche Tabou** : Utilise des structures de mémoire pour éviter de revisiter les mêmes solutions et pour explorer de nouvelles régions de l'espace de solution. Combine souvent des heuristiques d'amélioration comme 2-Opt pour raffiner les solutions.

3. Complexité

La complexité du TSP varie fortement selon l'approche utilisée :

- **Exacte** : O(n!), impraticable pour n>15 en général.
- **Heuristiques** : Complexité polynomial (souvent $O(n^2)$ ou O(n^3), adaptées pour des instances plus grandes mais sans garantie d'optimalité.
- **Métaheuristiques** : Complexité variable, généralement polynomiale, mais nécessitent souvent des réglages fins et peuvent être combinées avec des heuristiques pour des performances améliorées.

Conclusion

Le TSP est un problème classique en informatique et en optimisation, connu pour sa complexité intrinsèque. Les méthodes exactes sont limitées aux petits problèmes, tandis que les heuristiques et les métaheuristiques offrent des solutions viables pour des problèmes de taille pratique. L'utilisation de bibliothèques comme PuLP, numpy, scipy, et des solveurs comme CPLEX ou GUROBI facilite l'implémentation de ces méthodes en Python, permettant aux chercheurs et aux ingénieurs d'aborder efficacement ce problème complexe.

4 Implémentation

4.1 Dataset

Les jeux de données utilisés sont structurés sous forme de fichiers CSV, où chaque ligne représente une ville. Chaque ville est décrite par un objet avec les propriétés suivantes :

- Nombre de ville : c'est un entier.
- Temps d'exécution : Temps nécessaire pour effectuer la livraison.
- Longueur de la chaîne : Distance totale parcourue pour atteindre la ville.
- Type d'algorithme : Algorithme utilisé pour optimiser la tournée.

	Algorithm	Result	Execution Time	Nodes
1	Exact_PULP	226.898	4.474088907241821	8
2	Exact_GUROBI	226.89799997710995	0.5148756504058838	8
3	Exact_CPLEX	226.89800000000022	0.20816421508789062	8
4	H&M_Nearest Neighbor	234.182	0.0019958019256591797	8
5	H&M_Nearest Insertion	268.038	0.004984855651855469	8
6	H&M_Cheapest Insertion	281.843	0.004997968673706055	8
7	H&M_Farthest Insertion	261.166	0.005017757415771484	8
8	H&M_MST	254.998	0.0009996891021728516	8
9	H&M_2-Opt Exchange	134.247	0.9306762218475342	8
10	H&M_Tabu Search	134.24699999999999	0.1403636932373047	8

Figure 2 : Dataset

4.2 Implémentation de l'algorithme de résolution

Ce code implémente et compare différentes approches pour résoudre le problème du voyageur de commerce (TSP) en utilisant plusieurs heuristiques et méthodes exactes. Le code se décompose en plusieurs parties clés :

1. Génération de Problèmes TSP

La fonction generate_tsp_problem génère des matrices de distances aléatoires pour un nombre donné de villes et les enregistre dans des fichiers CSV. Cette étape crée des instances du problème TSP avec des distances symétriques et une diagonale nulle, nécessaires pour la résolution du problème.

```
def generate_tsp_problem(num_cities, filename):
    distances = np.random.rand(num_cities, num_cities) * 100  # Distances aléatoires entre 0 et 100
    distances = (distances + distances.T) / 2
    np.fill_diagonal(distances, 0)
    np.savetxt(filename, distances, delimiter=',', fmt='%.3f')

    print(f"Le problème TSP de taille {num_cities}x{num_cities} a été sauvegardé dans {filename}")
```

Figure 3 : Exemple de code

2. Classe tsp

La classe tsp contient plusieurs méthodes pour résoudre le problème TSP :

- **tsp_exact** : Utilise la programmation linéaire (avec les solveurs PULP, GUROBI, et CPLEX) pour trouver la solution exacte au problème TSP. Cette méthode est essentielle pour obtenir une référence optimale contre laquelle les heuristiques peuvent être comparées.
- **nn_heuristic, ni_heuristic, ci_heuristic, fi_heuristic, MST_heuristic** : Implémentent diverses heuristiques (proche voisin, insertion la moins chère, insertion la plus éloignée, arbre couvrant minimum) pour générer des solutions approximatives rapidement. Ces heuristiques sont des méthodes connues pour fournir des solutions rapides mais souvent sous-optimales.
- **Opt2, tabu_search** : Utilisent des techniques d'amélioration (2-opt et recherche tabou) pour améliorer les solutions heuristiques initiales. Ces méthodes permettent de raffiner les solutions obtenues par les heuristiques de base en réduisant les longueurs de trajet.

3. Exécution et Comparaison des Algorithmes

La boucle principale génère des problèmes TSP pour différentes tailles (10 à 20 villes), puis exécute chaque algorithme sur ces problèmes. Les résultats (routes trouvées, temps d'exécution, et coût total des trajets) sont stockés dans une liste et finalement dans un DataFrame pandas pour une analyse facile. La fonction run_algorithm mesure le temps d'exécution de chaque algorithme et formatte les résultats pour une comparaison cohérente.

4. Analyse des Résultats

Les résultats montrent les performances des différentes approches sur divers critères :

- **Temps d'exécution** : Les heuristiques sont rapides mais peuvent fournir des solutions moins optimales par rapport aux méthodes exactes, qui prennent plus de temps surtout pour des problèmes de plus grande taille.
- **Qualité des solutions** : Les méthodes exactes garantissent des solutions optimales tandis que les heuristiques offrent des compromis rapides. Les techniques d'amélioration comme le 2-opt et les recherches taboues tentent de rapprocher les solutions heuristiques de l'optimal.

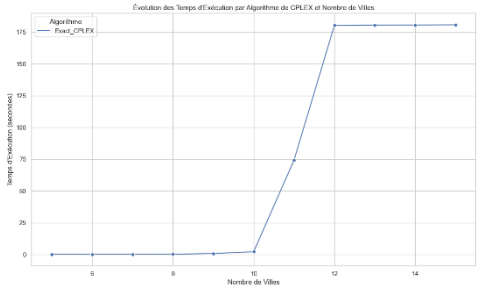
En résumé, ce code propose une approche exhaustive pour résoudre le TSP en combinant la rigueur des méthodes exactes avec la rapidité des heuristiques, tout en permettant une comparaison directe de leurs performances sur différents problèmes générés.

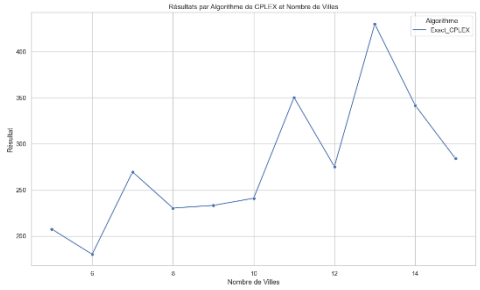
5 Etude expérimentale

5.1 Les algorithmes

5.1.1 Les solutions exactes

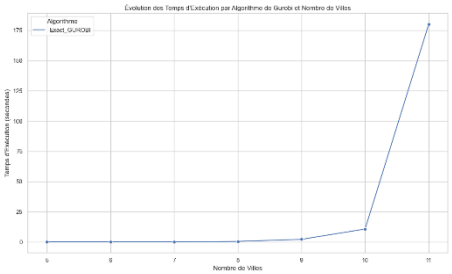
CPLEX :

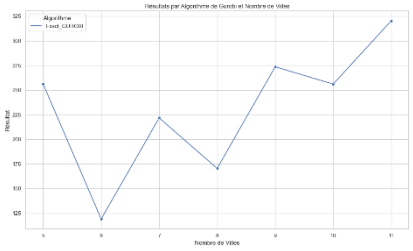




Nous pouvons observer que le solveur CPLEX suit une courbe exponentielle très pentue en termes de temps d'exécution à partir de 10 villes. De plus, il est important de noter que les solveurs sont soumis à une limite de temps maximale de 180 secondes (3 minutes). Il n'y a rien à signaler du point de vue de la résolution du problème les résultats obtenus sont cohérents.

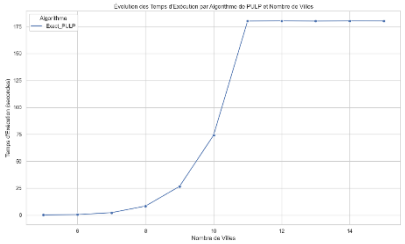
GUROBI

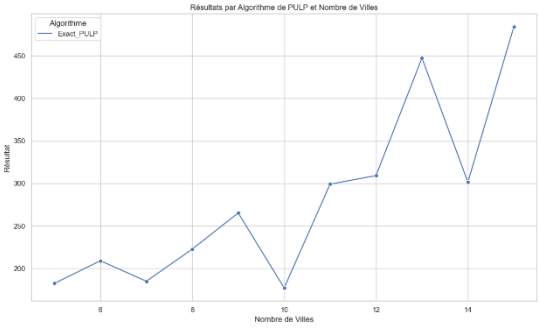




Ici aussi nous pouvons observer que le solveur GUROBI suit une courbe exponentielle très pentue en termes de temps d'exécution à partir de 10 villes. Il n'y a rien à signaler du point de vue de la résolution du problème les résultats obtenus sont cohérents.

Pulp

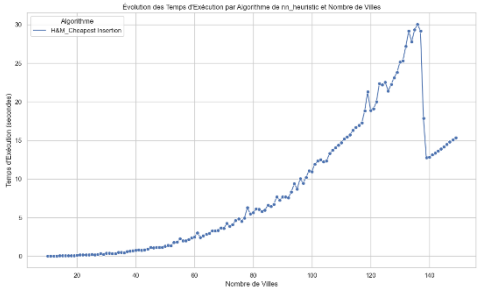


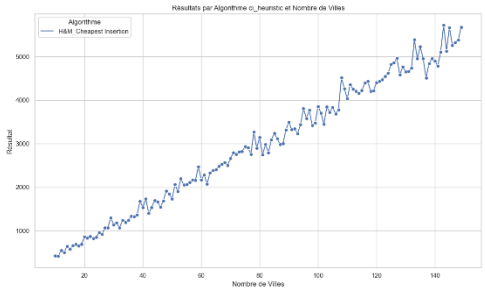


Ici aussi nous pouvons observer que le solveur PULP suit une courbe exponentielle moins pentue que les autres mais que celle-ci commence à partir de 8 villes. Il n'y a rien à signaler du point de vue de la résolution du problème les résultats obtenus sont cohérents.

5.1.2 Heuristics & Metaheuristics

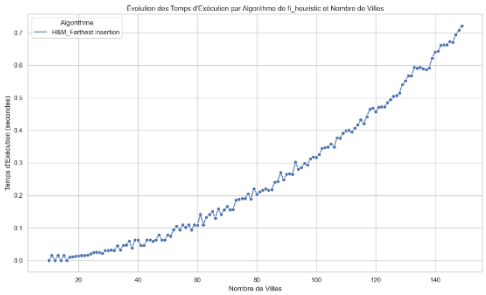
Cheapest intersection

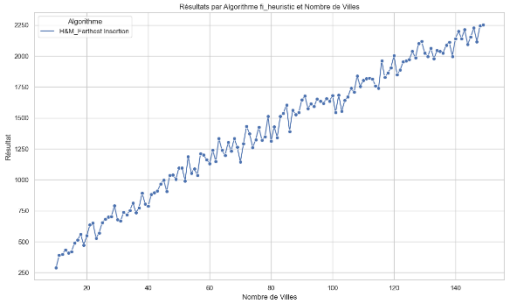




Les résultats obtenus sont cohérents, il forme logiquement une droite linéaire. On peut observer que le temps d'exécution lui suit une courbe exponentielle.

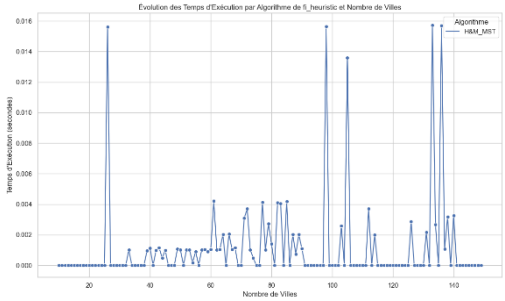
Farthest insertion

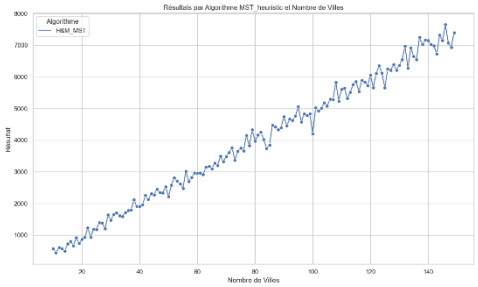




Les résultats obtenus sont cohérents, il forme logiquement une droite linéaire. On peut observer que le temps d'exécution lui suit une courbe très légèrement exponentielle.

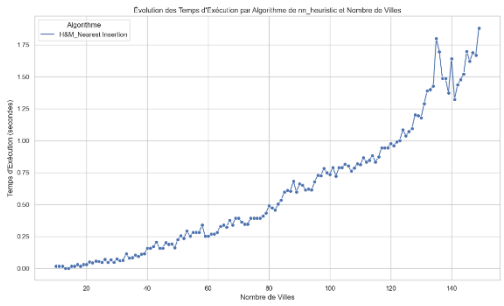
MST

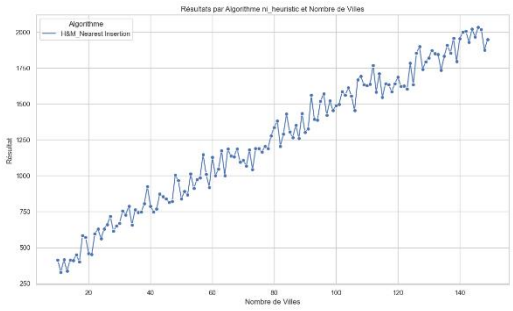




Les résultats obtenus sont cohérents, il forme logiquement une droite linéaire. On peut observer que le temps d'exécution lui suit une droite horizontale avec beaucoup d'interférence, il reste très bas.

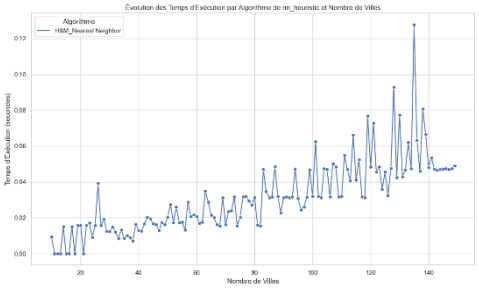
Nearest intersection

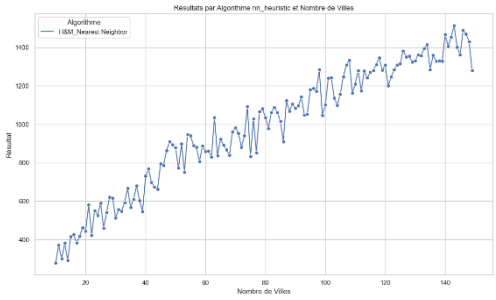




Les résultats obtenus sont cohérents, il forme logiquement une droite linéaire. On peut observer que le temps d'exécution lui suit une courbe très légèrement exponentielle.

Nearest Neighbors

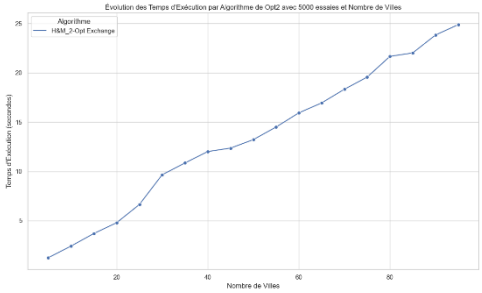


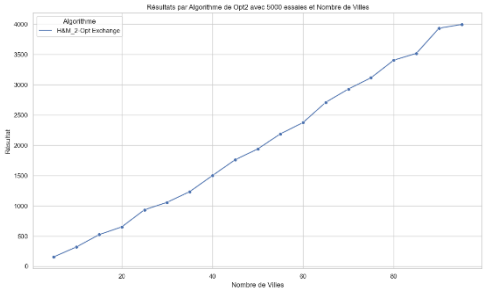


Les résultats obtenus sont cohérents, il forme logiquement une droite linéaire mais les solutions sont moins regroupées que les autres. On peut observer que le temps d'exécution lui suit une droite linéaire très dispersé.

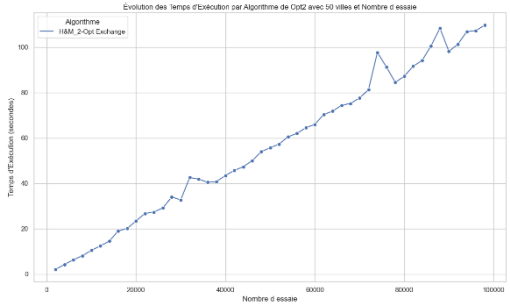
Algorithmes dynamiques

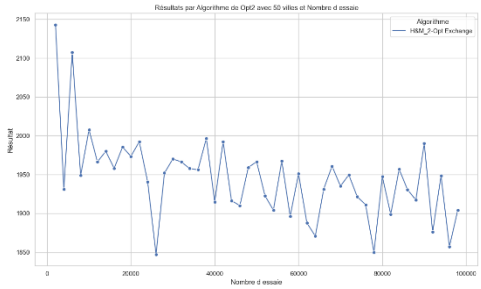
2-Opt Improvement





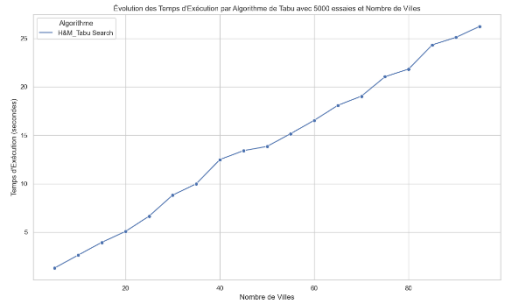
Dans ces deux premiers graphiques, nous testons l'évolution du temps d'exécution et les résultats obtenus avec un nombre d'essais fixe. On observe que les temps d'exécution suivent une tendance linéaire, tout comme les résultats, bien que ceux-ci ne soient pas les plus optimisés observés.

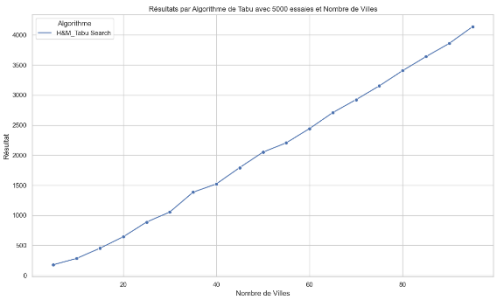




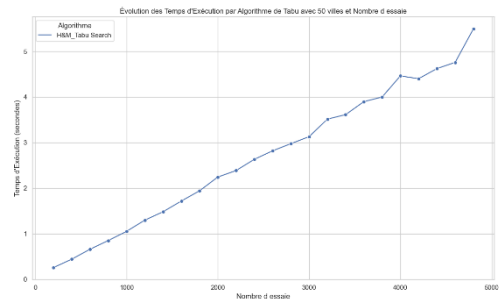
Dans ces deux graphiques suivants, nous testons l'évolution du temps d'exécution et les résultats obtenus avec un nombre de villes fixe mais un nombre d'essai variable. On observe que les temps d'exécution suivent une tendance linéaire et que les chemins obtenus avec plus d'essai sont de plus en plus court.

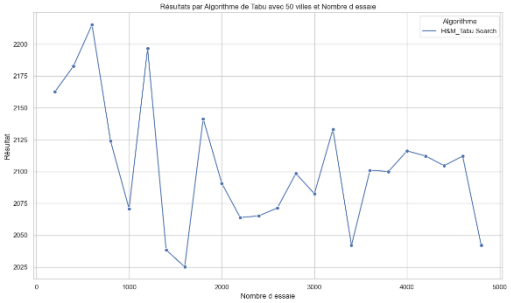
Tabu Search





Dans ces deux premiers graphiques, nous testons l'évolution du temps d'exécution et les résultats obtenus avec un nombre fixe d'essais. Nous observons que les temps d'exécution suivent une tendance linéaire, tout comme les résultats, bien que ces derniers ne soient pas les plus optimaux observés.



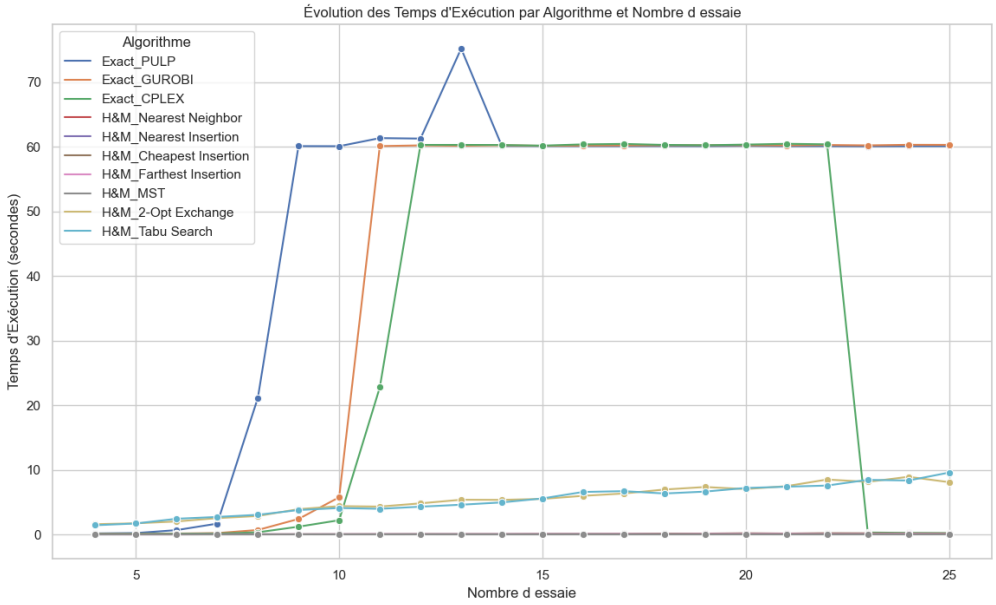


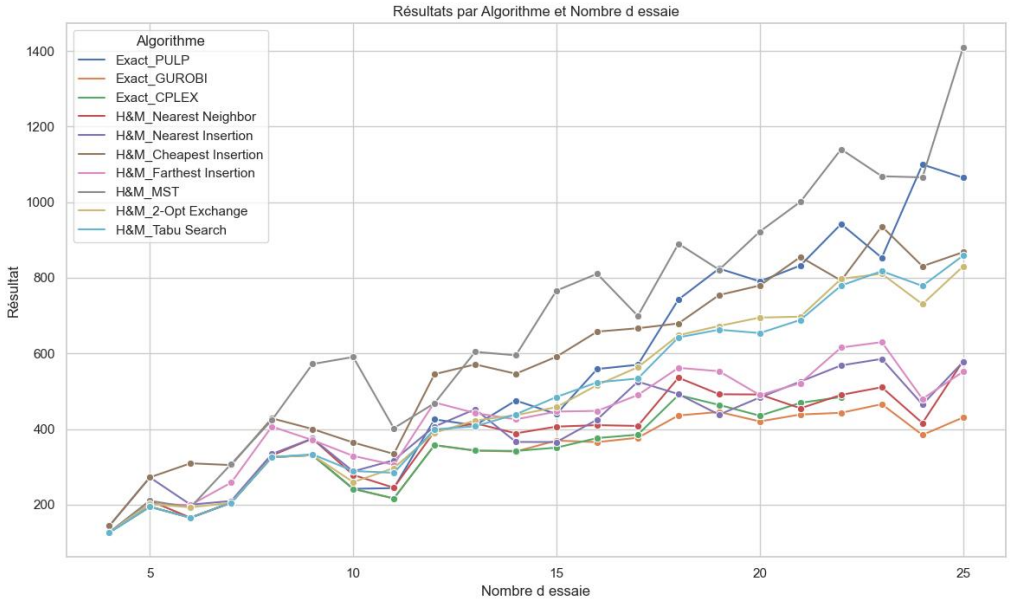
Dans ces deux graphiques suivants, nous testons l'évolution du temps d'exécution et les résultats obtenus avec un nombre fixe de villes mais un nombre variable d'essais. Nous

observons que les temps d'exécution suivent une tendance linéaire et que les chemins obtenus deviennent de plus en plus courts à mesure que le nombre d'essais augmente.

5.2 Comparaison

25 villes

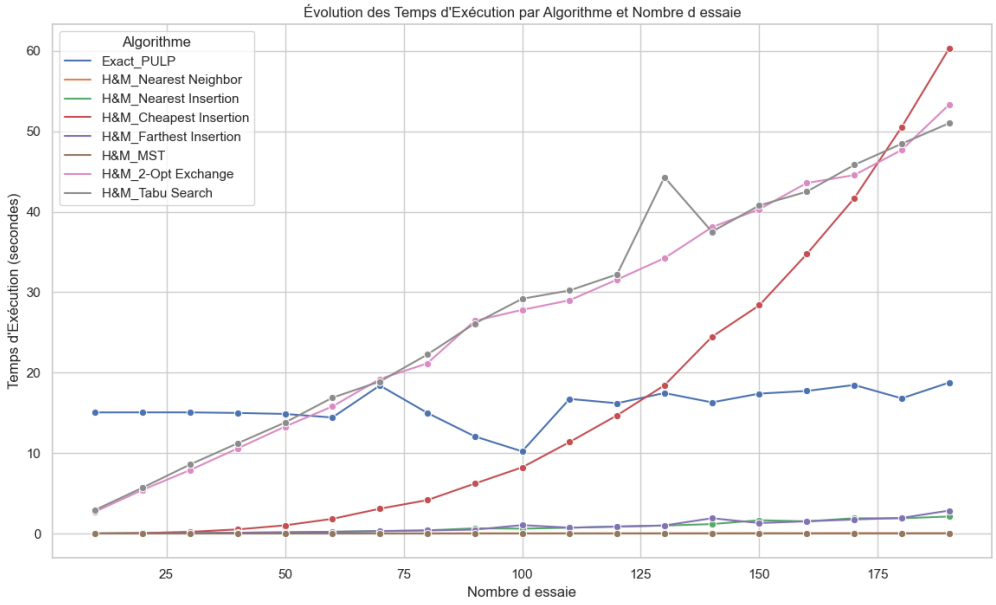


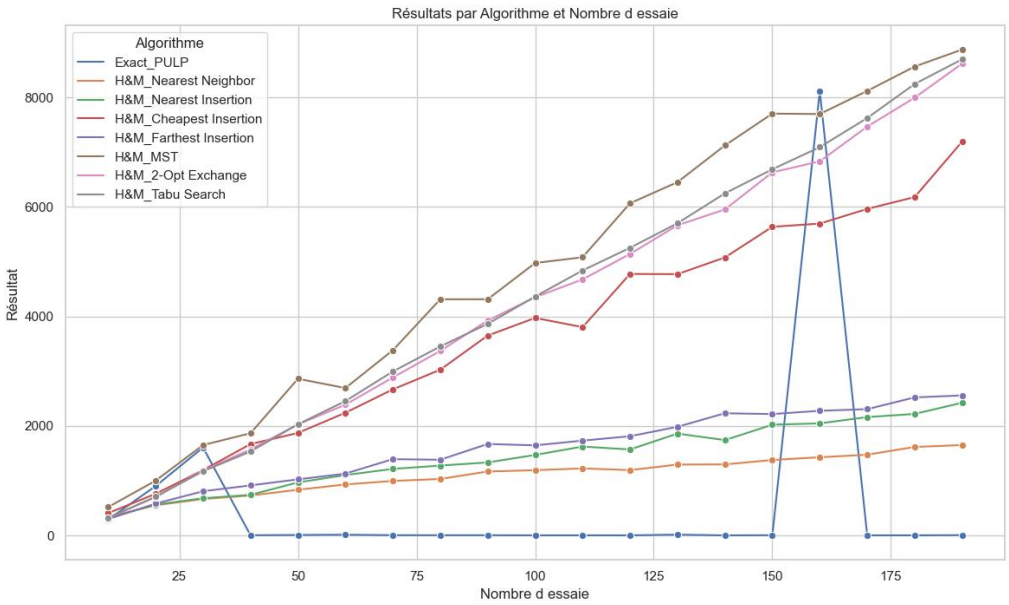


La première comparaison porte sur des instances allant de 4 à 25 villes, par incréments de 1, car nous sommes limités par les licences de GUROBI et CPLEX, qui ne nous permettent pas de résoudre des graphes de plus de 20 nœuds. Cela n'est pas problématique, car nous pouvons déjà observer que les temps d'exécution des solutions exactes sont longs, mais que les

résultats obtenus sont les plus courts. L'heuristique MST s'exécute le plus rapidement, mais produit les résultats les moins optimaux.

200 villes





Cette seconde comparaison porte sur des tests effectués avec un nombre de villes variant de 10 à 200 par incréments de 10. Nous observons que, au-delà de 30 villes, le solveur PuLP n'est plus capable de trouver une solution, probablement en raison de la limite de temps d'exécution fixée à 30 secondes. Il est toutefois remarquable qu'il ait réussi à trouver une solution pour 160 villes. Par ailleurs, nous constatons que l'algorithme du plus proche voisin (nearest neighbor) fournit les meilleurs résultats, et ce, dans l'un des temps d'exécution les

plus rapides. En revanche, l'algorithme de l'intersection la moins coûteuse (cheapest insertion) prend de plus en plus de temps à s'exécuter, jusqu'à devenir le plus lent, tout en produisant parmi les solutions les moins optimales.

6 Conclusion

Ce rapport présente une étude approfondie sur la modélisation et la résolution du Problème du Voyageur de Commerce (TSP). En réponse à l'appel à manifestation d'intérêt lancé par l'ADEME, l'initiative vise à développer des solutions de mobilité durable et efficace pour réduire l'impact environnemental et améliorer la gestion des ressources logistiques.

Nous avons commencé par un examen des défis logistiques liés au transport et leur importance pour des applications variées. Cela a été suivi par la modélisation du problème à l'aide de la théorie des graphes, permettant une représentation structurée des points de livraison et des routes. La modélisation linéaire et l'analyse de la complexité ont mis en lumière les contraintes et les difficultés associées à la résolution du TSP.

L'implémentation a été réalisée en Python, avec la génération de jeux de données et l'application de diverses heuristiques et méthodes exactes pour résoudre les instances du TSP. Les algorithmes utilisés incluent des heuristiques de construction (proche voisin, insertion la moins chère, etc.) et des techniques d'amélioration (2-opt, recherche taboue), ainsi que des méthodes exactes utilisant la programmation linéaire. Les résultats obtenus ont été analysés pour évaluer la qualité et l'efficacité des solutions en termes de distance totale parcourue.

L'étude expérimentale a montré que malgré l'augmentation du nombre de villes, l'algorithme parvient généralement à trouver des solutions efficaces et logiques. Les paramètres de l'instance du problème et ceux de l'algorithme influencent significativement le temps de convergence et la qualité des solutions, soulignant l'importance d'un ajustement soigneux de ces paramètres.

En conclusion, bien que l'algorithme présente une efficacité générale dans la résolution du TSP, il reste essentiel de continuer à optimiser les paramètres pour améliorer la qualité des solutions et le temps de calcul. Cette recherche ouvre la voie à des développements futurs pour créer des solutions de transport plus durables et intelligentes, alignées avec les objectifs de développement durable et d'innovation technologique de CesiCDP. La participation à ce projet stratégique offre une opportunité de contribuer à la transition vers des systèmes de mobilité plus écologiques et économiquement viables.