

Optimisation de livraison dans un réseau

Florent Tariolle et Amine Tauil

22 octobre 2025

1 Introduction

Ce document présente l'état de l'art concernant le problème d'optimisation de livraison dans un réseau urbain modélisé par un graphe. Le problème se décompose en deux étapes principales : le calcul des plus courts chemins entre tous les points d'intérêt, puis la résolution du problème du voyageur de commerce (TSP) pour déterminer l'itinéraire optimal de livraison.

2 Problématique

Le problème consiste à optimiser les livraisons depuis un dépôt vers plusieurs adresses dans une ville. La ville est modélisée par un graphe non orienté $G_1 = (V_1, E_1)$ où :

- V_1 représente les intersections, adresses de livraison et arrêts de métro
- E_1 représente les tronçons de route et les connexions du métro

L'objectif est de trouver un chemin optimal qui part du dépôt, visite toutes les adresses de livraison, et retourne au dépôt.

3 Approche en deux graphes

3.1 Premier graphe : Calcul des distances minimales

Le premier problème formel est le **All-Pairs Shortest Path (APSP)** : calculer la distance minimale entre chaque paire de sommets dans le graphe G_1 .

3.1.1 État de l'art pour APSP

Plusieurs algorithmes existent pour résoudre le problème APSP :

- **Algorithme de Floyd-Warshall** : complexité $O(V^3)$, adapté aux graphes denses
- **Repeated BFS** : complexité $O(V \times (V + E))$ pour les graphes non pondérés
- **Algorithme de Dijkstra répété** : complexité $O(V \times (E + V \log V))$ pour les graphes pondérés
- **Algorithmes avancés** : Seidel [1], techniques de multiplication de matrices rapides

3.1.2 Choix de l'algorithme : Repeated BFS

Nous avons choisi l'algorithme **Repeated BFS** (Breadth-First Search répété) pour les raisons suivantes :

1. **Simplicité d'implémentation** : Le BFS est un algorithme fondamental et facile à comprendre
2. **Efficacité pour graphes non pondérés** : Dans notre cas, chaque tronçon de route a une distance unitaire (1), ce qui rend le BFS optimal
3. **Complexité adaptée** : Pour un graphe avec V sommets et E arêtes, la complexité est $O(V \times (V + E))$, ce qui est efficace pour notre taille de graphe (22 sommets)
4. **Exactitude garantie** : Le BFS garantit de trouver le plus court chemin dans un graphe non pondéré

3.1.3 Pseudocode de Repeated BFS

Algorithm 1 Repeated BFS pour APSP

```

1: Initialiser une matrice  $V \times V$  de distances avec  $\infty$ , diagonale à 0
2: for chaque sommet source  $s$  dans le graphe do
3:   Initialiser une file avec  $s$  (distance 0), ensemble visité
4:   while file non vide do
5:     Défiler  $u$ 
6:     for chaque voisin  $v$  de  $u$  do
7:       if  $v$  non visité OU distance peut être mise à jour then
8:          $dist[s][v] = dist[s][u] + 1$ 
9:         Enfiler  $v$ , marquer comme visité
10:      end if
11:    end for
12:  end while
13: end for

```

3.2 Deuxième graphe : Graphe complet valué

À partir de la matrice des distances minimales, nous construisons un deuxième graphe $G_2 = (V_2, E_2)$ où :

- $V_2 = V_1$ (mêmes sommets que le graphe original)
- E_2 contient toutes les paires de sommets, chaque arête (u, v) étant valuée par la distance minimale entre u et v dans G_1

Ce graphe complet valué représente les distances optimales entre tous les points d'intérêt.

3.3 Problème du voyageur de commerce (TSP)

Sur le graphe complet G_2 , nous devons résoudre le TSP : trouver un cycle hamiltonien de coût minimal qui visite toutes les adresses de livraison en partant et revenant au dépôt.

3.3.1 État de l'art pour TSP

Le TSP est un problème NP-complet. Plusieurs approches existent :

- **Algorithmes exacts** :
 - **Held-Karp (programmation dynamique)** : complexité $O(2^n \times n^2)$ en temps, $O(2^n \times n)$ en espace
 - **Branch and Bound** : complexité exponentielle dans le pire cas
 - **Algorithmes de concorde** : optimisés pour des instances spécifiques
- **Algorithmes approchés** :
 - **Heuristiques constructives** : Nearest Neighbor, Insertion, etc.
 - **Amélioration locale** : 2-opt, 3-opt, Lin-Kernighan
 - **Métaheuristiques** : Recuit simulé, Algorithmes génétiques, Colonies de fourmis

3.3.2 Choix de l'algorithme : Held-Karp

Nous avons choisi l'algorithme de **Held-Karp** (aussi appelé algorithme de Bellman-Held-Karp) pour les raisons suivantes :

1. **Solution exacte** : Contrairement aux heuristiques, Held-Karp garantit la solution optimale
2. **Taille du problème gérable** : Avec 5 villes à visiter (dépôt + 4 adresses), la complexité $O(2^5 \times 5^2) = O(800)$ est instantanée
3. **Fiabilité** : Pas de dépendance aux heuristiques qui peuvent donner des résultats sous-optimaux

4. **Standard de l'industrie** : Held-Karp est l'algorithme standard pour les petites instances de TSP

3.3.3 Principe de Held-Karp

Held-Karp utilise la programmation dynamique avec des masques binaires :

- $dp[mask][j] = \text{coût minimal pour visiter toutes les villes dans } mask \text{ et finir à la ville } j$
 - Récurrence : $dp[mask][j] = \min_{k \in mask \setminus \{j\}} (dp[mask \setminus \{j\}][k] + dist(k, j))$
 - Cas de base : $dp[\{\}][] = 0$ où s est le dépôt
- La complexité est $O(2^n \times n^2)$ où n est le nombre de villes à visiter.

4 Justification des choix algorithmiques

4.1 Pourquoi Repeated BFS plutôt que Floyd-Warshall ?

- Notre graphe est **non pondéré** : toutes les arêtes ont un poids de 1
- Floyd-Warshall est conçu pour les graphes pondérés et nécessite $O(V^3)$ opérations
- BFS est spécialement optimisé pour les graphes non pondérés avec complexité $O(V + E)$ par source
- Pour notre graphe de 22 sommets, Repeated BFS est plus efficace et plus simple à implémenter

4.2 Pourquoi Held-Karp plutôt qu'une heuristique ?

- Le nombre de villes à visiter est **petit** (5 villes) : la complexité exponentielle est gérable
- Nous voulons la **solution optimale**, pas une approximation
- Les heuristiques peuvent donner des solutions jusqu'à 25% plus longues que l'optimum
- Held-Karp garantit la solution exacte avec un temps de calcul négligeable pour 5 villes

4.3 Pourquoi créer un graphe complet intermédiaire ?

- Le TSP nécessite les distances entre **toutes** les paires de villes
- Le graphe original contient des chemins indirects : créer le graphe complet pré-calcule tous les plus courts chemins

- Cela permet de résoudre le TSP sur un graphe complet, ce qui est l'hypothèse standard de Held-Karp
- La transformation est en $O(V^2)$ une fois que la matrice APSP est calculée

5 Complexité globale

- **Étape 1 - APSP** : $O(V \times (V + E)) = O(22 \times (22 + E))$ où E est le nombre d'arêtes
- **Étape 2 - Création du graphe complet** : $O(V^2)$
- **Étape 3 - TSP avec Held-Karp** : $O(2^n \times n^2)$ où n est le nombre de villes à visiter (5)
- **Complexité totale** : Dominée par l'APSP, donc $O(V \times (V + E))$

Pour notre instance avec 22 sommets et environ 29 arêtes (non orientées), la complexité est très raisonnable et le calcul est instantané.

6 Adaptation : Gestion de la panne du métro

6.1 Question théorique : Détection de l'utilisation du métro

Pour savoir si le trajet généré emprunte le métro ou non, il faut adapter la modélisation de la manière suivante :

1. **Marquer les arêtes du métro** : Attribuer un attribut ou un type différent aux arêtes correspondant aux connexions du métro (par exemple, ajouter un champ `type` aux arêtes : `ROUTE` ou `METRO`).
2. **Tracer le chemin dans le graphe original** : Après avoir résolu le TSP sur le graphe complet valué, reconstruire le chemin réel dans le graphe original G_1 . Pour chaque paire de villes consécutives dans la solution du TSP, trouver le plus court chemin dans G_1 qui les relie (utiliser un algorithme de reconstruction de chemin ou re-BFS).
3. **Vérifier si le chemin passe par des arrêts de métro** : Pour chaque segment du chemin reconstruit, vérifier s'il contient des arrêts de métro. Si le chemin entre deux adresses passe par `arret_metro_rouge`, `arret_metro_jaune` ou `arret_metro_vert`, alors le métro est utilisé.
4. **Afficher l'information** : Pour chaque étape du trajet, indiquer "Aller de X à Y via [MÉTRO / ROUTE]" et calculer le pourcentage du trajet effectué en métro.

En résumé : Il faut enrichir la modélisation pour distinguer les arêtes de métro, puis reconstruire le chemin dans le graphe original pour détecter quels segments utilisent effectivement le métro.

6.2 Impact de la panne du métro sur l'algorithme

6.2.1 Implications

La panne du métro implique les modifications suivantes :

- **Modification du graphe :** Supprimer les arêtes reliant les arrêts de métro entre eux (ligne de métro). Les arrêts de métro restent dans le graphe (accessibles par la route), mais ne sont plus reliés directement entre eux.
- **Impact sur le calcul des distances :** Le BFS répété recalcule automatiquement les distances sans les connexions du métro. Certaines distances entre arrêts de métro augmentent (passent de 1 à plusieurs tronçons). La matrice APSP change, notamment pour les paires impliquant des arrêts de métro.
- **Impact sur le graphe complet valué :** Les nouvelles distances sont reflétées dans le graphe complet. Le coût du trajet optimal peut augmenter et l'ordre des visites peut changer.
- **Impact sur Held-Karp :** L'algorithme Held-Karp fonctionne de la même manière, mais avec des distances différentes. La solution optimale peut changer en termes de coût et d'ordre des visites, mais reste exacte.

6.2.2 Modifications apportées au code

Pour gérer la panne du métro, nous avons modifié le code de la manière suivante :

1. **Ajout d'un paramètre à creerGrapheVille :**
 - Ajout d'un paramètre boolean metroActif pour contrôler si le métro est fonctionnel
 - Les arêtes du métro ne sont ajoutées que si `metroActif = true`
 - Les arrêts de métro restent accessibles uniquement par la route si le métro est en panne
2. **Création d'une méthode resoudreTSP :** Extraction de la logique de résolution du TSP dans une méthode réutilisable pour éviter la duplication de code.

3. **Modification du main** : Le programme résout maintenant le problème deux fois : une fois avec le métro fonctionnel et une fois sans le métro, permettant la comparaison des résultats.

6.3 Résultats comparatifs

6.3.1 Avec métro fonctionnel

- **Coût minimal** : 18
- **Chemin optimal** : depot_rue_bleu → 3_rue_marron → 22_rue_vert → 8_rue_mauve → 10_rue_rouge → depot_rue_bleu

6.3.2 Sans métro (panne)

- **Coût minimal** : 22
- **Chemin optimal** : depot_rue_bleu → 3_rue_marron → 22_rue_vert → 10_rue_rouge → 8_rue_mauve → depot_rue_bleu

6.3.3 Analyse

- **Impact sur le coût** : Le coût augmente de 18 à 22 (+22%), car sans le métro, certaines connexions directes entre arrêts de métro ne sont plus disponibles.
- **Impact sur l'ordre** : L'ordre des visites change : avec le métro, on visite 8_rue_mauve avant 10_rue_rouge, mais sans métro, l'ordre inverse devient optimal.
- **Robustesse** : Le graphe reste connexe car tous les arrêts de métro restent accessibles par la route, mais les distances augmentent.

7 Conclusion

L'approche en deux graphes avec Repeated BFS et Held-Karp est adaptée à notre problème car :

1. Elle exploite la structure du graphe (non pondéré) pour utiliser BFS de manière optimale
2. Elle garantit la solution optimale du TSP grâce à Held-Karp
3. Elle reste efficace pour la taille de notre instance (22 sommets, 5 villes à visiter)
4. Elle est simple à implémenter et à maintenir

5. Elle s'adapte facilement aux changements de contraintes (comme la panne du métro)

Cette combinaison d'algorithmes classiques et éprouvés constitue une solution robuste et exacte pour le problème d'optimisation de livraison.

Références

- [1] Seidel, R. (1992). On the all-pairs-shortest-path problem. *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pp. 745-749.
- [2] Held, M., & Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1), 196-210.
- [3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. Chapter 22 : Elementary Graph Algorithms.
- [4] Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2007). *The Traveling Salesman Problem : A Computational Study*. Princeton University Press.