Projet d'Optimisation

Le Problème du Voyageur de Commerce (TSP)

Rapport Final

Implémentation et Analyse d'Algorithmes

Master MIASHS — IMA-UCO (2025–2026)

Auteurs : Matthias Jourdren / Maxence Cornu Basset / Vincent Beausoleil / Abdenbi Iabbadene / Maxime De Ferry

Date : 13 janvier 2026

Table des matières

Introduction

Dans ce projet, nous nous intéressons au problème du Voyageur de Commerce (Traveling Salesperson Problem - TSP). L'objectif est de trouver le cycle hamiltonien de coût minimum dans un graphe complet pondéré. Nous avons implémenté et comparé quatre approches différentes :

— Une méthode exacte (Branch and Bound)
— Une heuristique constructive (Plus proche voisin)
— Une heuristique de recherche locale (2-opt)
— Une méta-heuristique (GRASP)

Méthodes Implémentées

Algorithme Exact : Branch and Bound

L'algorithme de séparation et évaluation (Branch and Bound) explore l'arbre des solutions possibles.

— **Principe** : Exploration en profondeur d'abord (DFS). On maintient le coût du meilleur tour trouvé ("upper bound"). Si le coût partiel d'un chemin dépasse ce coût, on coupe la branche ("pruning").
— **Complexité** : Dans le pire des cas, $O(n!)$, mais le pruning permet de réduire l'espace de recherche en pratique.

Algorithm 1: Pseudo-code Branch and Bound

```
Data: G : Graphe
Result: Meilleur tour, Meilleur coût
Fonction BranchAndBound(chemin_courant, cout_courant) :
if cout_courant ≥ meilleur_cout then
    Retourner
end
if tous les nœuds visités then
    cout_total ← cout_courant + cout(dernier, premier)
    if cout_total < meilleur_cout then
        meilleur_cout ← cout_total
        meilleur_tour ← chemin_courant
    end
    Retourner
end
foreach voisin non visité v do
    BranchAndBound(chemin_courant + v, cout_courant + cout(dernier, v))
end
```

Heuristique Constructive

Nous avons utilisé l'heuristique du **Plus Proche Voisin** (Nearest Neighbor).

— **Principe** : À chaque étape, on choisit la ville non visitée la plus proche de la ville courante.

— **Complexité** : $O(n^2)$.

Recherche Locale

Nous avons implémenté l'algorithme **2-opt**.

— **Principe** : On part d'une solution initiale (générée par l'heuristique constructive). On tente d'améliorer cette solution en inversant l'ordre de parcours entre deux villes i et k si cela réduit la distance totale.

— **Complexité** : Chaque itération prend $O(n^2)$.

Méta-heuristique GRASP

La procédure GRASP (Greedy Randomized Adaptive Search Procedure) combine une phase de construction aléatoire et une recherche locale.

— **Phase 1** : Construction gloutonne randomisée. On construit une liste restreinte de candidats (RCL) contenant les meilleures villes suivantes (selon un paramètre α). On choisit aléatoirement une ville dans cette liste.

— **Phase 2** : On applique la recherche locale (2-opt) sur la solution construite.

— Ces deux phases sont répétées un certain nombre de fois et on garde la meilleure solution globale.

Expérimentations et Résultats

Protocole

Les tests ont été effectués sur un ordinateur [Spécifications]. Nous avons utilisé des instances de taille variable pour comparer les temps d'exécution et la qualité des solutions.

Comparaison des Performances

Conclusion