Le problème du voyageur de commerce
Comparaison de la complexité de plusieurs algorithmes
Projet à rendre le 7 novembre 2025

Présentation du Projet

Le problème du voyageur de commerce (TSP - Travelling Salesman Problem) est l'un des problèmes les plus célèbres en informatique théorique et en recherche opérationnelle. Étant NP-complet, il sert de benchmark pour évaluer l'efficacité des différentes approches algorithmiques.

Dans ce projet, vous implémenterez et comparerez quatre approches différentes pour résoudre le TSP sur des instances de tailles variées, en utilisant un notebook Jupyter comme support unique de rendu.

Rendu : Ce projet est à réaliser **seul** ou en **binôme**.

Le rendu doit se faire avec **un seul** notebook qui contiendra le code dans des cellules *code* et le texte dans des cellules *markdown*.

Si vous développez beaucoup de code (création de classes, fonctions utilitaires, fonctions graphiques, etc.) vous pouvez alléger le notebook en mettant une partie de ce code dans un fichier python *.py que vous importerez depuis le notebook. Vous pouvez aussi ajouter des fichiers de données.

Le nom du fichier (des fichiers si vous importez une partie de votre code ou ajoutez des fichiers de données) à rendre doit être de la forme

```
projet_NOM1_Prénom1[_NOM2_Prenom2].ipynb
[projet_NOM1_Prénom1[_NOM2_Prenom2].py]
[Data1.dat]
....
[DataN.dat]
```

Entre `[...]`, les parties optionnelles.

Algorithmes à Implémenter

1. Générateur d'instance de problèmes TSP
2. Algorithme Force Brute (BF-TSP) : Approche exhaustive qui évalue tous les cycles hamiltoniens possibles.
   Complexité théorique : $O(n!)$
3. l'algorithme de Held-Karp : Algorithme utilisant la programmation dynamique.
   Complexité théorique : $O(n^2 \cdot 2^n)$
4. L'algorithme de Christofides et Serdyukov : un algorithme basé sur les arbres recouvrants et les cycles eulériens.
   Complexité théorique : $O(n^3)$
5. Algorithmes de recherche locale
   — Recherche locale avec opérateur de voisinage k-opt :
   — **2-opt** : Échange de deux arêtes pour réduire la longueur du cycle
   — **3-opt** : Échange de trois arêtes pour une amélioration plus significative
   — **Recuit simulé** : Algorithme méta-heuristique inspiré de la thermodynamique. Il utilise un opérateur de voisinage (typiquement 2-opt) et accepte occasionnellement des solutions moins bonnes.
   Complexité : Dépend des paramètres (nombre d'itérations)

Pour illustrer ces algorithmes, vous utiliserez des instances TSP de différentes tailles :
— Instances de test (5-10 villes)
— Instances petites (10-15 villes)
— Instances moyennes (15-20 villes)
— Instances grandes (20+ villes) - pour les algorithmes approchés seulement

Attendus : Le but de ce projet est de
- **Comprendre** la complexité algorithmique du problème TSP
- **Implémenter** différentes stratégies de résolution
- **Analyser** empiriquement les performances des algorithmes
- **Comparer** les compromis entre qualité de solution et temps d'exécution
- **Produire** des visualisations claires et informatives

Pour chaque algorithme et instance, les métriques à évaluer sont :
- Le nombre d'itérations/opérations (complexité théorique)
- Le temps d'exécution **moyen** (complexité empirique)
- La qualité de la solution (longueur du cycle)
- L'utilisation mémoire (optionnel)

Barème : Votre projet sera évalué sur 100 points. Le barème sera le suivant :
- La partie code ne rapporte **rien** ! (0 points).
  Remarque : Vous trouverez du code python "prêt à l'emploi" sur internet. Vous pouvez aussi utiliser une IA générative pour produire le code. Dans tous les cas, validez le code.
- Il sera tenu compte de la qualité du notebook : lisibilité du code, qualité des commentaires et des illustrations (10 points).
- Le début du notebook doit impérativement comporter une brève présentation de la théorie de la complexité (10 points).
- Chaque algorithme doit être décrit de manière **concise** et sa complexité doit être étudiée de manière théorique (30 points).
- La partie la plus importante de votre rendu sera la comparaison de tous les algorithmes selon différents critères (40 points) :
  - temps d'exécution (à mettre en lien avec la complexité),
  - qualité de la solution (pour les algorithmes approchés),
  - compromis temps-qualité de la solution (pour les algorithmes approchés).

  Vous illustrerez vos comparaisons en produisant les courbes d'évolution de ces critères lorsque le nombre d'instances augmente.
- Vous terminerez en faisant une conclusion générale et un compte rendu des apports de ce projet [analyse réflexive] à votre compréhension de la science informatique (10 points).

Références Bibliographiques

1. Cormen, T. H., et al. *Introduction to Algorithms*. MIT Press.
2. Johnson, D.S. and McGeoch, L.A. (1997), *The Traveling Salesman Problem : A Case Study in Local Optimization*. Local Search in Combinatorial Optimization, 1, 215-310.
3. *TSPLIB* : `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`
4. *Blog* : `https://github.com/sandipan/Blogs/blob/master/2020-11-19_Travelling-Salesman-Problem--TSP--w`
   `md`