Projet Final
Problème du Voyageur de Commerce

Introduction

Le Problème du Voyageur de Commerce (TSP) pose la question suivante : « Étant donnée une liste de villes et les distances entre chaque paire de villes, quel est le plus court trajet possible qui visite chaque ville et revient à la ville d'origine ? » Il s'agit d'un problème NP-difficile en optimisation combinatoire, important en recherche opérationnelle et en informatique théorique.

Le problème a été formulé pour la première fois en 1930 et est l'un des problèmes les plus étudiés en optimisation. Il est utilisé comme benchmark pour de nombreuses méthodes d'optimisation. Bien que le problème soit computationnellement difficile, de nombreuses heuristiques et algorithmes exacts sont connus, de sorte que certaines instances avec des dizaines de milliers de villes peuvent être résolues complètement et même des problèmes avec des millions de villes peuvent être approximés à une fraction de 1%.

Formellement, soit $G = (V, E, w)$ un graphe complet non orienté tel que $|V| = n$ et $w : E \to \mathbb{R}_{\geq 0}$ la fonction de poids. C'est-à-dire que G est le graphe K_n avec des poids sur ses arêtes. Dans ce modèle, chaque sommet représente une ville et chaque arête pondérée reliant deux sommets représente la distance entre les villes correspondantes.

Ainsi, étant donné un graphe complet pondéré non orienté, le TSP consiste à trouver un cycle hamiltonien (un cycle qui traverse chaque sommet exactement une fois) qui minimise la somme de ses poids.

Exercices

1. Décrire (d'autres) situations réelles qui peuvent être modélisées comme un TSP.
2. Développer et implémenter un algorithme exact pour résoudre le TSP, utilisant la technique du Branch and Bound.
3. Développer et implémenter une heuristique constructive pour résoudre le TSP.
4. Développer et implémenter une heuristique de recherche locale pour résoudre le TSP.
5. Développer et implémenter un algorithme utilisant la méta-heuristique GRASP [1] pour le TSP. À chaque itération du GRASP, utiliser comme première phase une modification de l'heuristique donnée en 3, et comme seconde phase l'heuristique donnée en 4. **FIXER** tous les paramètres impliqués (taille de la liste restreinte de candidats (RLC), critères d'arrêt, etc.) via des expériences. **JUSTIFIER** vos choix et le choix des instances de test utilisées pour fixer vos paramètres.
6. Pour chacune des méthodes 2 à 5 :
   - Expliquer en détail votre algorithme. **NE PAS** donner le code, utiliser du pseudo-code à la place.
   - Calculer sa complexité temporelle en utilisant le pseudo-code donné précédemment.
   - Décrire des instances du TSP pour lesquelles la méthode ne fournit pas de solution optimale. À quel point la solution obtenue peut-elle être mauvaise par rapport à la solution optimale ?
   - Faire des expériences permettant d'observer la performance de l'algorithme. Comparer le temps d'exécution avec sa complexité théorique. Comparer également la qualité des solutions obtenues et le temps d'exécution en fonction de la taille de l'entrée (et d'autres paramètres si approprié). Les cas de test doivent inclure, comme cas pathologiques, ceux décrits dans l'item précédent. Dans le cas où l'algorithme a des paramètres configurables déterminant son comportement (la méta-heuristique par exemple, bien qu'il soit ouvert aux autres également), vous devez exécuter des expériences en faisant varier les valeurs des paramètres et choisir, si possible,

les réglages offrant les meilleurs résultats pour les instances utilisées. Présenter les résultats obtenus à l'aide de graphiques appropriés.

7. Une fois que vous avez choisi les meilleures valeurs de configuration pour chaque heuristique, exécuter des expériences sur un **nouvel ensemble d'instances** pour observer les performances des méthodes et comparer à nouveau la qualité des solutions obtenues et le temps d'exécution. Présenter tous les résultats obtenus en utilisant des graphiques appropriés et présenter des conclusions sur votre travail.

Directives générales

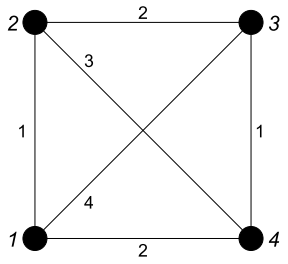
1. Un rapport couvrant tous les points décrits ci-dessus doit être rédigé en format PDF. L'utilisation de `LaTeX` [2] est fortement recommandée.
2. Vous pouvez utiliser n'importe quel langage de programmation pour implémenter vos algorithmes, **MAIS** utilisez le même pour tous. Lors de l'envoi de votre projet, vous devez inclure dans le code source un fichier appelé "`README.md`" qui explique comment compiler/exécuter le code.
3. Vos algorithmes doivent lire un fichier d'entrée (voir l'exemple ci-dessous) contenant $n + 1$ lignes comme suit :
   - La première ligne contient le nombre n de sommets.
   - Les n lignes suivantes contiennent les lignes de la matrice d'adjacence (avec les poids séparés par un espace) du graphe complet. Notez que la matrice est symétrique.
4. Vos algorithmes doivent écrire un fichier contenant la solution (voir l'exemple ci-dessous) comme suit :
   - Le nom du fichier doit être "`instance_method.out`" où "`instance`" est le nom du fichier d'entrée (sans extension) et "`method`" est l'algorithme parmi "`exact`", "`constructive`", "`local_search`" et "`grasp`".
   - Le fichier doit contenir deux lignes. La première contient les numéros des sommets séparés par un espace, représentant l'ordre dans lequel les villes doivent être visitées. La seconde ligne doit contenir le poids du cycle.
5. Vous devez également fournir les instances de test que vous avez utilisées pour fixer les paramètres configurables et l'ensemble final d'instances utilisé pour comparer les performances (car ils doivent être référencés dans le rapport).
6. **TRÈS IMPORTANT :** Pendant l'exécution des expériences mesurées,
   - **NE FAITES** rien d'autre avec votre ordinateur **ET** fermez tous les programmes inutiles (comme chrome, firefox, etc.). Ne pas le faire peut affecter la performance de vos algorithmes et donner des résultats incorrects. **NE** branchez/débranchez **PAS** non plus votre ordinateur.
   - **UTILISEZ** le même ordinateur (et dans les mêmes conditions spécifiées ci-dessus) pour comparer les temps d'exécution. Il est insensé (donc cela donnera de mauvais résultats) de comparer les temps d'exécution, par exemple, d'un algorithme A contre un algorithme B sur deux ordinateurs différents.

Exemple

Considérons le fichier "`test.in`" :

```
4
0 1 3 2
1 0 2 4
3 2 0 1
2 4 1 0
```

Ce fichier correspond au graphe suivant :



Ainsi, si nous exécutons l'algorithme exact sur cette instance, le fichier "`test_exact.out`" sera créé et contiendra uniquement les deux lignes suivantes.

```
1 2 3 4
6
```

Cela correspond au cycle $1, 2, 3, 4, 1$ avec un poids total $1 + 2 + 1 + 2 = 6$.

Deadline & instructions pour le rendu

La date limite du projet est le xxxxx XX xxxxxxx 2026, xxh, sur XXXXX.
AUCUN FICHIER NE SERA ACCEPTÉ APRÈS LA DATE LIMITE.

Vous devez téléverser uniquement **UN** fichier compressé nommé "`team_X.zip`", où "`X`" est le numéro de votre équipe. Un seul membre de l'équipe doit téléverser le fichier. Ce fichier doit contenir les éléments suivants :

- Un fichier nommé "`README.md`" qui explique comment les dossiers/fichiers du projet sont organisés, comment compiler/exécuter le code source, où les fichiers exécutables seront créés, où les fichiers d'entrée doivent être placés, où les fichiers de sortie seront écrits et toutes les instructions nécessaires.
- Un dossier nommé "`report`" contenant le fichier pdf nommé "`report_team_X.pdf`" de votre rapport et ses fichiers sources.
- Un dossier nommé "`src`" contenant à l'intérieur les sous-dossiers suivants :
  i. Sous-dossier "`model`" : contenant le code source de votre modèle de graphe et les fonctions de base partagées par tous les algorithmes.
  ii. Un sous-dossier distinct pour chaque algorithme nommé "`method`", où "`method`" doit être "`exact`", "`constructive`", "`local_search`" et "`tabu_search`". Chacun de ces dossiers doit contenir le code source de l'algorithme correspondant.
- Un dossier nommé "`instances`" : contenant un sous-dossier distinct pour chacun de vos algorithmes nommé "`method`", où "`method`" sera "`exact`", "`constructive`", "`local_search`" et "`tabu_search`". Chacun de ces dossiers doit contenir les instances que vous avez utilisées pour tester vos algorithmes individuellement et/ou pour régler les paramètres configurables (qui doivent être référencés dans le rapport). Il doit également contenir un dossier appelé "`new_instances`" contenant le **nouveau jeu d'instances** que vous avez utilisé pour comparer les performances entre les algorithmes comme décrit précédemment. Vous devez compresser tous les fichiers d'instances si leur taille est importante.

Si vous utilisez `github` ou tout autre gestionnaire de dépôt, **SUPPRIMEZ** tous les fichiers et dossiers cachés que le gestionnaire crée. **Ne pas le faire entraînera une pénalité de points**.

Références

[1] Thomas A. Feo and Mauricio G. C. Resende. *Greedy randomized adaptive search procedures*. Journal of Global Optimization, pages 1–27, 1995.

[2] *An introduction to LaTeX*. LaTeX project (`https://www.latex-project.org/about/`).