Rapport du projet 1 de théorie des graphes

Maxence Ahlouche Maxime Arthaud Korantin Auguste
Martin Carton Thomas Forgione Thomas Wagner

todo date

1 TPs

Nous avons commencé les TPs par le choix du langage utilisé pour implémenter nos algorithmes : le python, (pour la simplicité qu'il offre pour représenter les graphes et la lecture/le parsage d'un fichier représentant un graphe), puis l'écriture des classes permettant de représenter un graphe.

Puis nous nous sommes séparés en trois binômes ayant travaillé sur :

— un algorithme permettant de savoir si un graphe est connexe ou non (dans le cas non orienté) ;
— la création d'un algorithme (brute force) qui trouve un chemin hamiltonien dans un graphe ;
— la création de quelques fonctions qui génèrent des graphes simples de tests.

2 Problème 1

2.1 Exercice 1

Pour représenter ce problème par un graphe, on représente les carrefours par les nœuds du graphe et les routes par des arêtes. Goudronner toutes les routes revient alors à parcourir toutes les arêtes une et une seule fois dans le graphe.

Afin que le problème soit soluble, il faut que le graphe soit eulérien, c'est à dire connexe et que chaque nœud ait un nombre pair d'arêtes. Cette condition n'est pas nécessaire, car il peut toutefois aussi contenir exactement deux nœuds ayant un nombre impair d'arêtes, si on part d'un de ces deux nœuds (et on arrivera à l'autre).

On peut utiliser l'algorithme d'Euler, qui consiste à trouver un cycle dans le graphe (en le parcourant « au hasard »), puis à s'appeler récursivement sur le sous-graphe construit en retirant les arêtes du cycle trouvé, en partant d'un nœud du cycle qui n'a pas d'arêtes qui n'appartiennent pas à ce cycle.

2.2 Exercice 2

On commence par construire le graphe représentant le musée où les arêtes représentent les portes et les sommets des salles du musée. Par définition, l'objectif de cet enfant est réalisable si ce graphe est hamiltonien ou semi-hamiltonien.

Un graphe hamiltonien est un graphe qui contient au moins un cycle hamiltonien, c'est à dire un cycle passant une et une seule fois par chaque sommet en revenant au sommet de départ. S'il est uniquement semi-hamiltonien, il n'a qu'une chaîne hamiltonienne : l'enfant ne se retrouvera pas dans la salle de départ.

Un moyen simple de résoudre ce problème est de tester toutes les possibilités de chaînes, jusqu'à en trouver une qui soit hamiltonienne. Toutefois, cette solution a une complexité en $O(n!)$ (avec n le nombre d'arêtes), et par conséquent est peu envisageable pour des graphes de grande taille.

Nous avons implémenté cet algorithme dans la fonction `hamiltonian_path`.

Pour améliorer cet algorithme, on peut arrêter de chercher quand le graphe restant à parcourir n'est pas connexe.

Un autre moyen serait de calculer les puissances successives de la matrice latine représentant le graphe. Il suffirait alors d'éliminer tous les chemins ne contenant pas une et une seule fois tous les sommets du graphe. Lors du calcul des puissances successives, on peut remplacer par des 0 tous les chemins contenant deux fois le même sommet. Toutefois, cet algorithme est également très complexe.

Ce problème étant NP-complet, il n'existe pas de moyen simple de déterminer le chemin, ni même si un tel chemin existe pour un graphe donné quelconque.

2.3 Exercice 3

Il est possible de résoudre le premier problème en transformant le graphe : il suffit de créer le graphe dont les sommets sont les arêtes du premier graphe, et les arêtes sont les sommets de ce premier graphe. Ce nouveau graphe est appelé « line graph ». En trouvant une chaîne hamiltonienne dans ce graphe, on obtient une chaîne eulérienne.

Par contre, il est impossible de transformer le problème 2 en problème 1. Le problème 2 est NP-complet, alors que le premier est soluble en $O(n)$ (à l'aide par exemple de l'algorithme de Hierholzer). Les 2 problèmes ne sont donc pas équivalents.

3 Problème 2

3.1 Exercice 1

Si le graphe est eulérien, ou semi-eulérien, l'algorithme d'Euler résout le problème, en trouvant un cycle qui passe par toutes les routes, une et une seule fois : c'est donc forcément la solution la plus efficace.

Sinon, il suffit de transformer le graphe en un graphe eulérien en créant des arêtes : on prend le graphe partiel contenant uniquement les sommets de degré impair, qu'on transforme en clique. Pour cela, pour chaque couple de sommets non reliés entre eux, on

crée une arête les rejoignant, de poids égal au coût le plus faible possible pour rejoindre ces sommets sur le graphe total.

Puis on cherche le couplage parfait de coût minimum : c'est à dire l'ensemble des arêtes disjointes couvrant tous les sommets du graphe (il existe car on aura forcément un nombre de sommets pair dans le graphe partiel).

Il suffit de doubler le nombre des arêtes de cet ensemble, on obtient ainsi un graphe eulérien, et avec l'algorithme d'Euler, on a un parcours passant par toutes les arêtes, qui est donc optimal (cf. le cas du graphe eulérien ci-dessus).

3.2 Exercice 2

Pour résoudre ce problème, on commence par construire le graphe où les sommets sont les points à percer et où on relie tout les sommets par des arêtes (le graphe est complet), auquel on attribue un poids correspondant à la distance entre ces points.

Le problème est alors un problème du voyageur de commerce dans ce graphe : on cherche une chaine hamiltonienne de poids minimal, dont on sait qu'il en existe au moins une (car le graphe est complet).

Ce problème est NP-difficile, on ne connait donc pas d'algorithme efficace pour résoudre ce problème.

Pour résoudre ce problème, nous avons implémenté l'algorithme "du plus proche voisin", cette algorithme est un algorithme approximatif, dont on sait qu'il peut donner le pire chemin sur certains graphes.

Nous avons lancé cet algorithme sur plusieurs "grands" graphes [1], les résultats sont présentés dans la table 1 (sont données les longueurs des chemins trouvés et le ratio longueur trouvée/longueur minimale).

Fichier de test	Meilleur résultat	Plus proche voisin
berlin52.tsp	7542	8981, 119%
bier127.tsp	118282	139505, 118%
d657.tsp	48912	62176, 127%
fl1577.tsp	22249	N/A [2]
u724.tsp	41910	55344, 132%

Table 1 – Résultats pour TSP

On remarque que bien qu'il ne fournisse aucune garantie, l'algorithme du plus proche voisin donne des résultats plutôt bons.

1. Trouvés sur `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

3.3 Exercice 3

Ce problème ressemble au problème précédent sans la contrainte de ne passer qu'une seule fois par ville/point à percer. On pourrait donc utiliser les mêmes algorithmes, mais il pourrait y avoir des solutions plus efficaces en repassant par une ville déjà visitée.