RAPPORT PROJET : VOYAGEUR DE COMMERCE

Réalisé par Caitlin Dagg, Manon Pintault et Benjamin Saint-Sever.

Sommaire

RAPPORT

Introduction

SUJET DU PROJET

Le but de ce projet était de fournir un programme fonctionnant en ligne de commande permettant de calculer des solutions (pas forcément optimales) au problème du voyageur de commerce métrique (c'est-à-dire calculer le meilleur trajet à parcourir pour un ensemble de villes données sans repasser par une ville déjà visitée). Dans le cadre de ce projet, l'ensemble des villes a été donnée sous forme de matrice de distances.

DEPOT PRINCIPAL

Afin de pouvoir réaliser ce projet, nous avons choisi d'utiliser un dépôt en ligne : github. Si nous avons opté pour cette solution c'est car il nous permettait de choisir entre l'utilisation de svn ou git. De plus c'est un outil simple d'utilisation et accessible avec une simple connexion internet depuis n'importe quel support (mac, windows, linux).

ORGANISATION

Pendant toute la durée du projet, nous nous sommes partagés les tâches chaque semaine. Par exemple, il y en avait un qui faisait un algorithme, un autre s'occupait du makefile, de réfléchir sur le prochain algorithme à faire et le dernier créait des tests et gérait l'organisation des fichiers. Mais généralement nous rajoutions chacun des lignes de codes aux différents algorithmes et modules pour améliorer ceux-ci, ajouter des fonctionnalités et régler quelques soucis rencontrés.

Une heuristique simple: Nearest Neighbour

L'algorithme Nearest Neighbour consiste à visiter le point le plus proche non-visité jusqu'à obtenir un parcours complet des points. Cette algorithme ne fournit que rarement le parcours optimal, mais a un complexité en O(n) (contrairement au O(n !) de l'algorithme de brute force), et fournit un distance qui se situe à environ deux fois la distance minimale, ce qui peut le rendre utile.

Notre algorithme prends en paramètre un matrice de points, et partant du premier, utilise la matrice des distances et la propriété 'visitéé/non-visitée' du structure de donnée des points pour trouver le prochain point. Ce point est ensuite ajouté au liste des points representant le parcours, et on recommence en cherchant le « nearest neighbour » de ce point.

Un algorithme d'approximation: Prim

L’algorithme de Prim, utilise le principe de Minimum Spanning Tree (MST), et consiste à essayer de connecter tous les points, ou nœuds, en suivant les vertices ayant un coût minimum.

Un algorithme exact par recherche exhaustive: Brute Force

L'algorithme de Brute Force trouve la solution du problème de voyageur de commerce en calculant la distance totale de chaque parcours possible, et retourn la solution optimale.

Notre fonction Brute Force fonctionne en trouvant chaque permutation possible d'un liste de points (en utilisant du swapping et backtracking), en calculant la distance totale avec la fonction overallDistance, et enfin comparant avec la valeur minimum actuel – si la valeur est plus petite que cette minimum, alors on prends cette valeur comme nouvel minimum, et copie la permutation actuel des points vers la liste « pOut ».

La « vraie » fonction Brute Force, appelé « bruteForceRough » est récursif : il prends donc plusieurs paramètres, certains assez obscurs. Pour faciliter l'utilisation de cette fonction, on a crée un function « wrapper » (appelé « bruteForce»), qui prend comme unique paramètre une matrice et retourne une liste de points, et fournit au fonction récursif les paramètres nécessaires.

Cette solution est lente, de complexité O(n !), et n'est donc utilisable en pratique que dans des cas où n est très petit.

Un algorithme exact: Branch and Bound

L'algorithme Branch and Bound, comme son nom l'indique, est base deux principes: branching, et bounding (séparation et évaluation).

- Branching : on pourrait représenter l'ensemble des solutions du TSP par un arbre (que ce soit planaire, avec chaque nœud représentant les points qu'on peut visiter, ou binaire, avec un nœud représentant l'inclusion d'un point dans le parcours, et l'autre son exclusion). La branching consistera alors à choisir le meilleur nœud, en fonction d'un critère donné.
- Bounding : c'est avec le bounding qu'on choisit sur quel parcours continuer. Dans notre algorithme, ceci consiste à calculer la « lower bound », c'est-à-dire la moins grande distance possible d'un parcours donné.

En combinant ces deux principes, on crée donc un algorithme en partant du même principe que brute force, mais qui, en pratique, diminue fortement le temps de recherche de la solution.

Conclusion

Nous avons pu mettre en pratique des savoirs acquis au fils des autres cours, par exemple : l'algorithme de permutation utilise des concepts vus en cours d'algorithme de S3, le découpage modulaire vu en cours de programmation aussi du semestre dernier, et l'utilisation de valgrind et des makefile vu en cours d'EDD ce semestre.

Néanmoins, nous conseillons aux voyageurs de commerce de contacter Montgomery Scott au lieu des étudiants de L2, ce serait peut-être plus simple…