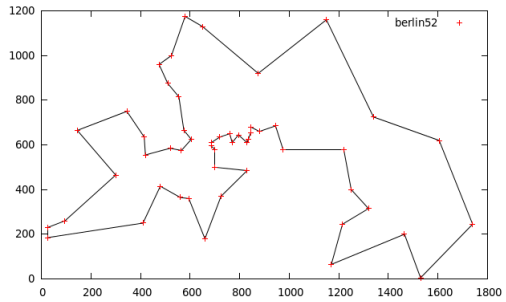
Mini-projet - ROB3 2018-2019
Autour du problème de voyageur de commerce

Le sujet au format pdf ainsi que les instances sont disponibles sur le site de l'UE :
`https://moodle-sciences.upmc.fr/moodle-2018/course/view.php?id=2370`

Le problème du voyageur de commerce se formule comme suit. Un voyageur de commerce doit visiter un ensemble prédéfini de villes. Partant de sa ville de résidence, il réalise une tournée où chaque ville est visitée exactement une fois avant de finalement rentrer à sa ville de résidence (cycle Hamiltonien). Connaissant les distances entre les villes, on cherche à déterminer la meilleure séquence de visites, de façon à minimiser la longueur totale. Ce problème est certainement le plus connu des problèmes d'optimisation combinatoire. Sa résolution est requise pour de nombreuses applications (distribution du courrier, ramassage des déchets ménagers, etc.). Il a été très largement étudié, et les nombreux travaux menés ont souvent conduit à l'introduction de nouvelles techniques d'optimisation combinatoire, qui ont ensuite dépassées le cadre de ce problème.
Dans ce projet, nous nous intéressons à la version Euclidienne du problème, où l'on dispose des coordonnées d'un ensemble de points dans le plan, et où l'on cherche une tournée de longueur minimale au sens de la somme des distances Euclidiennes entre les points. Voici ci-dessous un exemple de tournée optimale pour 52 points dans la ville de Berlin.



Dans ce mini-projet, on va implémenter une méthode pour calculer une borne inférieure de la valeur d'une solution optimale, ainsi qu'une méthode pour déterminer une solution approchée. Pour visualiser les tournées obtenues et aider au débogage, on pourra faire appel aux fonctions et procédures d'affichage déjà implantées dans le fichier *projet.c* (téléchargeable depuis le site web de l'UE). Une bibliothèque d'instances est disponible dans l'archive *instances.zip*. Dans cette archive, les fichiers .tsp comportent les coordonnées des points, et les fichiers .opt.tour comportent les tournées optimales lorsque celles-ci sont connues.

Question 0 - Parcourir le fichier *projet.c* pour prendre connaissance des fonctions et procédures d'affichage déjà implantées.

1 Calcul d'une borne inférieure sur la longueur d'une tournée optimale

Pour calculer une borne inférieure, on aura besoin de résoudre le problème de l'arbre couvrant de valeur minimum sur le graphe obtenu en définissant un sommet pour chaque point dans le plan, une arête pour chaque couple de points, et la valeur d'une arête comme la distance Euclidienne entre les deux points. Afin de limiter les ressources nécessaires en espace mémoire, on ne stockera pas explicitement le graphe en mémoire mais on calculera à la demande la valeur d'une arête en calculant la distance Euclidienne entre ses deux extrémités.

Question 1 - Si on fait appel à un tas pour stocker les sommets de la bordure (voir les transparents de cours), quelle est la complexité de l'algorithme de Prim en fonction du nombre n de points ?

Question 2 - Coder l'algorithme, puis le tester sur quelques instances de la bibliothèque.

On appelle 1-arbre un graphe tel que le sous-graphe sur $\{2, \ldots, n\}$ est un arbre (graphe connexe sans cycle), et qui comporte de plus deux arêtes incidentes à 1. Compte tenu de cette définition, étant donné un graphe, un graphe partiel qui est un 1-arbre de valeur minimum peut être obtenu en utilisant l'algorithme de Prim pour déterminer un arbre couvrant minimum sur le sous-graphe sur les sommets $\{2, \ldots, n\}$, puis en ajoutant à cet arbre deux arêtes incidentes au sommet 1 de valeur minimum.

Question 3 - Prouver que la valeur d'un 1-arbre optimal est une borne inférieure de la longueur d'une tournée optimale.

Question 4 - Coder un algorithme qui retourne un 1-arbre de valeur minimum, puis le tester sur quelques instances de la bibliothèque.

Afin de raffiner cette borne inférieure, supposons maintenant qu'on assigne un poids $\pi(s)$ à chaque sommet s du graphe $G = (S, A)$, et que l'on définit la valeur $w_\pi(\{s, s'\})$ d'une arête $\{s, s'\}$ par : $w_\pi(\{s, s'\}) = d(s, s') + \pi(s) + \pi(s')$, où $d(s, s')$ la distance Euclidienne entre les deux points correspondant.

Question 5 - Prouver qu'une tournée optimale pour le problème du voyageur de commerce sur le graphe G muni des valeurs $d(s, s')$ le reste sur le graphe G muni des valeurs $w_\pi(\{s, s'\})$, et inversement.

Question 6 - Prouver que :

$$\min_{T \text{ un 1-arbre}} w_\pi(T) - 2\sum_{s \in S} \pi(s) \leq \min_{C \text{ un cycle Hamiltonien}} d(C)$$

où $w_\pi(T) = \sum_{\{s,s'\} \in T} w_\pi(\{s, s'\})$ et $d(C) = \sum_{\{s,s'\} \in C} d(s, s')$.

Pour tout jeu de poids π, $\min_{T \text{ un 1-arbre}} w_\pi(T) - 2\sum_{s \in S} \pi(s)$ est donc une borne inférieure de la valeur d'une tournée optimale.

Question 7 - Coder un algorithme qui prend en argument le jeu de poids π et retourne la borne inférieure correspondante, puis le tester sur quelques instances de la bibliothèque.

La question précédente fournit une famille de bornes inférieures (une borne inférieure par jeu de poids π, borne inférieure obtenue par le calcul d'un 1-arbre de valeur minimum). On vise maintenant à obtenir un jeu de poids π tel que la borne inférieure soit la plus grande possible (et donc la plus précise possible). Soit T_π le 1-arbre minimum obtenu pour les valeurs w_π sur les arêtes. Il s'agit de déterminer un jeu de poids π tel que T_π ressemble le plus possible à une tournée, c'est-à-dire que les degrés des sommets dans T_π soient le plus proche possible de 2. Pour ce faire, on va procéder par une méthode itérative, où l'on définit π_{i+1} en fonction de π_i au terme de chaque itération i en fonction du 1-arbre T_{π_i} obtenu. Soit $d_i(s)$ le degré du sommet s dans T_{π_i}. Si $d_i(s) > 2$ (resp. $d_i(s) < 2$), alors on définit $\pi_{i+1}(s) > \pi_i(s)$ (resp. $\pi_{i+1}(s) < \pi_i(s)$) de façon à faire diminuer (resp. augmenter) le degré de s dans T_π. Plus précisément, on utilise les formules suivantes pour mettre à jour les jeux de poids :

$$\begin{cases} \pi_0(s) &= 0 \qquad \forall s \\ \pi_{i+1}(s) &= \pi_i(s) + t_i(d_i(s) - 2) \end{cases} \qquad \text{avec } t_i = \frac{\lambda_i(\text{UB} - w_{\pi_i}(T_{\pi_i}))}{\sum_{s \in S}(d_i(s) - 2)^2}$$

où UB est un majorant de la valeur de la solution optimale. On prend comme majorant UB la valeur indiquée dans le fichier *valeursUB.txt* (majorant propre à chaque instance). Soit n le nombre de sommets de G. On pose $\lambda_i = 2$ pour $2n$ itérations, puis $\lambda_i = 1$ pour n itérations, puis $\lambda_i = 1/2$ pour $n/2$ itérations, puis $\lambda_i = 1/2^2$ pour $n/2^2$ itérations, etc. On s'arrête lorsque $n/2^k < 1$. On retourne alors la valeur de la borne inférieure calculée à la dernière itération.

Question 8 - Implémenter cet algorithme, puis comparer expérimentalement sur diverses instances de la bibliothèque les temps de calcul et les bornes inférieures obtenues pour cet algorithme comparativement à l'algorithme de la question 4.

2 Un algorithme glouton et un algorithme de recherche locale

On s'intéresse dans cette section à déterminer une solution approchée du problème de voyageur de commerce. On se propose tout d'abord d'employer l'algorithme "glouton" suivant. On part d'une tournée vide et on trie les arêtes du graphe par valeurs croissantes. En considérant les arêtes dans cet ordre croissant, on insère une arête dans la tournée en construction dès lors qu'elle ne crée pas de cycle vis-à-vis des arêtes déjà insérées dans la tournée et que les sommets aux extrêmités sont toutes les deux de degré strictement inférieur à 2 (dans le graphe partiel sur les arêtes déjà insérées). On arrête l'algorithme dès qu'on a inséré $n-1$ arêtes. On "boucle" alors la tournée en insérant l'arête reliant les deux sommets de degré 1 dans le graphe partiel.

Question 9 - Si on fait appel à une structure de données *union-find* pour tester si une arête crée un cycle (voir les transparents de cours), quelle est la complexité de l'algorithme en fonction de n ?

Question 10 - Coder l'algorithme, puis le tester sur diverses instances de la bibliothèque, en rapportant les temps de calcul et en comparant la valeur de la tournée obtenue avec celle de la borne inférieure de la question 8.

On se propose d'améliorer la solution approchée obtenue à la question précédente en procédant de la manière suivante. On considère le cycle Hamiltonien C trouvé par l'algorithme glouton. A toute paire d'arêtes $\{s, s'\}$ et $\{s'', s'''\}$ appartenant à cette tournée on fait correspondre la tournée C' définie par : $C' = C \cup \{\{s, s''\}, \{s', s'''\}\}$. L'échange est réalisé si la longueur de C' est inférieure à celle de C. Il s'agit d'un 2-échange. Une tournée est dite localement optimale pour 2-opt si tout 2-échange conduit à dégrader la valeur de la tournée. On réitère cette recherche locale jusqu'à obtenir une tournée localement optimale pour 2-opt.

Question 11 - Quelle est la complexité d'une itération de cet algorithme ? Peut-on en conclure que la complexité de l'algorithme est polynomiale ?

Question 12 - Coder l'algorithme, puis le tester sur diverses instances de la bibliothèque, en rapportant les temps de calcul et en comparant la valeur de la tournée obtenue avec celle de la borne inférieure de la question 8.

Question ouverte (bonus) - Proposer des améliorations à l'algorithme de la question 12.

3 Travail demandé

Le travail se fait obligatoirement en binôme du même groupe de TP. Il compte pour 25% de la note finale du module. Les algorithmes sont à implémenter en C.

Rapport. Chaque binôme doit rédiger sur traitement de texte un rapport (de 5 pages environ, et dans tous les cas moins de 10 pages) répondant aux différentes questions de l'énoncé (quelques explications et commentaires peuvent ne pas être superflus).
Vous créerez un répertoire ayant pour nom `nomBinome1_nomBinome2`, où `nomBinome1` et `nomBinome2` correspondent à vos noms de famille. Ce répertoire contiendra votre rapport (au format pdf) ainsi que les fichiers sources *commentés* de vos programmes. Compressez ce répertoire sous forme d'un fichier `zip`. Envoyez ce fichier en pièce attachée d'un courrier électronique ayant pour sujet `[rob3-algorithmique] MiniProjet`, et adressé à votre chargé de TP (`thibaut.lust@lip6.fr` ou `hugo.martin@lip6.fr` selon votre groupe), au plus tard le **jeudi 7 mars 2019** (23:59:59).

Soutenance. La soutenance a lieu lors de la dernière séance de TP (semaine du 11 mars). Un planning des passages vous sera communiqué. Lors de la soutenance, vous ferez une présentation de 10 minutes (**avec transparents**, directement sur machine) pour résumer votre travail, en veillant à bien vous partager le temps de parole, puis 10 minutes seront consacrés à l'examen interactif du code réalisé.

NB : il pourra être demandé d'exécuter le code sur des instances autres que celle du projet. Votre programme devra donc pouvoir résoudre des instances fournies sur des fichiers au même format que ceux de la bibliothèque.