Projet du cours *Informatique pour le Génie Industriel*
Autour du problème du voyageur de commerce

École des Mines de Nancy 2A

Frédéric Sur
frederic.sur@loria.fr
www.loria.fr/~sur/enseignement/projetIGI2007/

1 Contexte

Le problème dit *du voyageur de commerce* (Traveling Salesman Problem) est un grand classique de l'optimisation combinatoire. Sa formulation est très simple : étant données n villes, trouver un chemin de longueur minimale passant une et une seule fois par chacune des n villes et revenant au point de départ. Dans la suite, on appellera un tel chemin passant une seule fois par chacune des villes et revenant au point de départ un *circuit*.

Le problème de décision associé (étant donné un entier N, existe-t-il un circuit de longueur inférieure à N ?) étant NP-complet, il est peu probable qu'on puisse trouver un algorithme « efficace » (polynomial en n) pour ce problème[1].

Dans ce projet on explorera différentes heuristiques pour trouver des solutions approchées. Le but ici n'est pas d'examiner des méthodes s'appuyant sur des résultats plus forts qui permettent par exemple de mesurer si la longueur du circuit trouvé est éloignée de la longueur optimale (inconnue). Néanmoins, vous pouvez proposer de tels algorithmes pour la section 9.

2 Modalités pratiques du projet

- Travail de programmation VB.net personnel ou par groupe de deux, au choix (merci de me préciser le choix et la composition des groupes dès les premières séances).
- Évaluation sur la base d'un rapport écrit de quelques pages, décrivant les algorithmes étudiés et la modélisation adoptée. Vous comparerez les performances relatives des différents algorithmes (en terme de longueur du circuit pour la solution trouvée et de temps de recherche de cette solution) et veillerez à présenter clairement vos conclusions. N'attendez pas la fin du projet et rédigez au fur et à mesure de la progression. Une démonstration sur machine sera également à prévoir.

3 Travail de programmation en Visual Basic.net

Vous implanterez les algorithmes des sections 4 (résolution exacte) et 5 à 9 (heuristiques) en langage VB.net. Dans un premier temps, vous testerez vos algorithmes sur un jeu de données synthétiques avec un faible nombre de villes (huit villes est raisonnable), que vous aurez généré vous-même. Ensuite, vous travaillerez sur les jeux de données de la section 10.

[1] Voir la liste des « Millenium Problems » à un million de dollars : `http://www.claymath.org/millennium/P_vs_NP/`

Étape 1 (Construction de l'interface graphique)

Vous pouvez commencer par construire une interface graphique avec des boutons pour chacun des algorithmes et des fenêtres pour visualiser les données ainsi que la solution trouvée.

Il sera intéressant de représenter graphiquement les villes à relier, ainsi que le chemin calculé à chaque itération des algorithmes.

Il faudra prévoir une procédure d'ouverture du fichier de données.

4 Résolution exacte

Dans le cas où il y a peu de villes à visiter, on peut envisager de chercher les solutions optimales de manière exhaustive. Ceci fournit un élément de comparaison pour évaluer la performance des heuristiques qui seront implantées par la suite.

Quel est le nombre de circuits possibles entre n villes ? S'il faut 1ms pour calculer la longueur d'un circuit, combien de temps la recherche exhaustive prend-elle pour 10 villes ? 20 ? 50 ?

Étape 2 (Résolution exacte)

Implantez cette recherche exhaustive pour n villes, et testez avec le jeu de données de huit villes.

Indication. Tout le problème est de générer l'ensemble des circuits possibles sur n villes. Supposons qu'on décrive ces circuits sous la forme $(1, x_2, \dots x_n)$, où $x_2, \dots, x_n$ sont les numéros de villes en position $2, \dots, n$.[2] Remarquons que l'on peut ramener ce problème à celui de générer les $(n-1)!$ permutations des éléments $2, \dots, n$. Voici un algorithme qui donne le successeur d'une permutation donnée lorsque celles-ci sont classées dans l'ordre lexicographique.

Entrée : une liste $L = (x_1, x_2, \dots x_n)$ représentant une permutation.

Sortie : une nouvelle liste L' représentant une permutation, successeur de la liste L dans l'ordre lexicographique.

Trouver, en parcourant la liste « de droite à gauche » (par indices décroissants) le plus grand $i < n$ tel que x_i est plus petit que $\max_{i<j\leq n} x_j$. On note i_0 cet indice.

Trouver x_{j_0} tel que x_{j_0} est le plus petit x_j ($i_0 < j \leq n$) supérieur à x_{i_0}

Construire $L' = (x_1, \dots, x_{i_0-1}, x_{j_0}, \mathcal{L})$, où $\mathcal{L}$ est la liste formée des éléments non déjà placés dans L', classés dans l'ordre croissant.

Exemple. Pour $n = 4$, le successeur de $(1, 3, 4, 2)$ est $(1, 4, 2, 3)$.

En initialisant au circuit $(1, 2, 3, \dots, n)$ et en appliquant de manière itérative l'algorithme trouvant le successeur jusqu'à atteindre le circuit $1, n, n-1, \dots, 2$, on parcourt l'ensemble des circuits possibles.

Exemple. Pour $n = 4$, vérifiez qu'en partant de $(1, 2, 3, 4)$ on décrit l'ensemble des permutations par applications successives de l'algorithme.

Une autre manière de faire est décrite ici : `http://en.wikipedia.org/wiki/Permutation`
Vous pouvez choisir d'implanter plutôt cet algorithme.

5 Algorithmes gloutons

5.1 Un premier algorithme naïf

Une heuristique naïve pour la construction d'une solution pourrait être : on choisit une ville, puis on va à la ville la plus proche.

Étape 3 (Heuristique naïve)

Implantez cet algorithme.

[2] Comme les circuits sont fermés, on peut toujours commencer à les décrire en commençant par une ville fixée.

Exhibez un exemple où cet algorithme ne fournit pas une solution optimale.

5.2 Algorithmes par insertion

On peut envisager des algorithmes par insertion construisant le circuit des villes de manière itérative. Si on a construit un circuit de i villes à visiter, on insère dans ce circuit une $i+1$-ème ville (choisie parmi celles qui ne sont pas encore dans le circuit).

Étape 4 (Insertion)
Testez les trois possibilités d'insertions suivantes :

1. dans le cas où les villes sont ordonnées *a priori*, insérer la $i+1$-ème ville dans la liste de manière à minimiser la longueur du parcours ;
2. insérer la ville la plus proche d'une des villes de la liste de manière à minimiser la longueur du parcours total ;
3. insérer la ville la plus éloignée du circuit (i.e. la ville maximisant la distance minimale à une ville du circuit).

D'autres heuristiques par insertion (ou des hybridations de ces heuristiques) sont possibles, n'hésitez pas à les programmer.
Pour cette question, il sera peut-être intéressant d'implanter le circuit à construire sous forme d'`Arraylist`[3].

6 Optimisation locale

Voyez-vous une manière immédiate de réduire la longueur d'un circuit qui présente un « croisement » (i.e. deux arcs entre villes qui se couperaient) ? Au passage, ceci serait-il valable si l'on cherchait à minimiser le coût de transport (avec des péages éventuels, une consommation d'essence différente selon le type de voie, etc) plutôt que la distance kilométrique totale parcourue ?

Étape 5 (Optimisation locale)
Proposez un algorithme exploitant cette remarque.

Indication. Il s'agit de tester de manière itérative si transformer deux arcs (x_{i_0}, x_{i_1}) et (x_{j_0}, x_{j_1}) en (x_{i_0}, x_{j_0}) et (x_{i_1}, x_{j_1}) diminue la longueur totale du circuit.

7 Recuit simulé et algorithmes génétiques

Le recuit simulé et les algorithmes génétiques sont des classiques de l'optimisation combinatoire, que vous verrez en cours.

Étape 6 (Recuit simulé)
Proposez une formulation du recuit simulé pour ce problème, et implantez-là. Vous testerez différentes méthodes d'initialisation de la recherche, et l'influence des paramètres de l'algorithme.

Étape 7 (Algorithme génétique)
Proposez des règles de reproduction, hybridations, mutations pour un algorithme génétique, et implantez l'algorithme génétique correspondant. On testera différents types de règles.

[3] Consultez par exemple la base de connaissance Microsoft : `http://msdn2.microsoft.com/fr-fr/default.aspx`

8 Procédure de séparation et évaluation (PSE) « bonus »

Les heuristiques précédentes vous ont fourni un circuit peut-être assez proche de l'optimal. En organisant l'ensemble des circuits possibles en arbre (chaque nœud correspond à un sous-circuit, d'un niveau à l'autre on ajoute une ville au circuit, et les circuits complets sont les feuilles de l'arbre), on peut envisager une procédure de séparation et évaluation (*branch and bound*). On explore l'arbre par un parcours en profondeur en arrêtant l'exploration d'une branche lorsque la longueur du sous-circuit correspondant est déjà plus grande que celle de l'optimum courant.

On pourra s'inspirer de l'algorithme permettant de parcourir un ensemble de permutations décrit dans la section 4.

Étape 8 (Séparation et évaluation)
Implantez la procédure PSE, et faites des expériences. Cela vous permet-il d'atteindre le circuit de longueur minimale sur les jeux de données fournis ?

9 Question ouverte « bonus »

Étape 9 (Initiative personnelle)
Testez un algorithme que vous aurez trouvé dans la littérature sur le sujet, et comparez-le aux algorithmes précédents.

10 Jeux de données

De nombreux jeux de données (ainsi que de la lecture intéressante) se trouvent sur la page web suivante : `http://www.tsp.gatech.edu/world/countries.html`

Vous essaierez vos algorithmes sur `Djibouti` et `Luxembourg` qui ont l'avantage d'être de taille limitée. Ces fichiers figurent aussi sur la page web du projet.