Université Paul Sabatier
TOULOUSE III

Licence Informatique
Projet avancé

Problème du voyageur de commerce appliqué, entre autres, à l'art de faire des trous

Vincent Dugat - Septembre 2025

Table des matières

Le sujet est composé de plusieurs parties. Après une présentation générale du problème, vous trouverez différentes sections présentant chacune une méthode de résolution et suivies d'une partie "implémentation" qui précise ce qu'il faut programmer.

Prenez le temps de bien lire ces parties. Le respect des spécifications sera pris en compte dans l'évaluation.

Les différentes parties partagent les structures de données et un certain nombre de sous-programmes. La structure et la cohérence globale du projet fera partie de l'évaluation finale.

Les différentes parties donneront lieu à des dépôts sur Moodle et des présentations en présentiel.

1 Présentation du problème

Etant donnés n points (qu'on peut appeler villes, sommets, ou noeuds) répartis sur un plan, et les distances les séparant, trouver un chemin de longueur totale minimale qui passe exactement une fois par chaque point et revienne au point de départ (une tournée). En effet, selon l'ordre dans lequel on visite les villes, on ne parcourt pas la même distance totale. C'est un problème d'optimisation combinatoire qui consiste à trouver la meilleure solution parmi un ensemble de choix possibles.

Il est clair que toutes les tournées (liste ordonnée des n points), sont des solutions du problème. Ce qu'on veut c'est la solution minimale en terme de somme totale des distances.

Ce problème a des applications en informatique industrielle et peut servir tel quel à l'optimisation de trajectoires de machines-outils : par exemple, pour minimiser le temps total que met une fraiseuse à commande numérique pour percer n points dans une plaque de tôle ou pour percer les trous des composants d'un circuit électronique (motherboard), l'optimisation des VLSI, ou en recherche opérationnelle (recherche de chemins minimaux, gestion de transport), etc.

Ce problème est plus compliqué qu'il n'y paraît et on ne connaît pas de méthode de résolution permettant d'obtenir des solutions exactes en un temps raisonnable (polynomial) pour de grandes instances (grand nombre de villes) du problème. En effet ce problème est NP-dur. Pour ces grandes instances, on devra donc souvent se contenter de solutions approchées, car on se retrouve face à une explosion combinatoire : le nombre de chemins possibles passant par 69 villes est déjà un nombre d'une longueur de 100 chiffres.

Le problème du "voyageur de commerce" a été étudié depuis longtemps et on dispose d'une grande variété d'algorithmes donnant le plus souvent des solutions approchées mais calculables en un temps raisonnable.

Entendons-nous sur le vocabulaire à l'aide de quelques définitions :

Définition : *Un graphe complet K_N est un graphe avec N sommets et une arête entre tous les couples de sommets possibles.*

Définition : *Un cycle hamiltonien est un cycle qui passe une fois et une seule par chaque sommet du graphe.*

Définition : *Un graphe pondéré est un graphe où toutes les arêtes possèdent un coefficient numérique appelé poids (temps, distance, coût, etc.).*

Définition : *Le problème du voyageur de commerce (Traveling Salesman Problem aka TSP) est le problème de trouver le cycle hamiltonien de poids minimal de K_N.*

Remarque : Nous nous limitons aux problèmes de TSP symétriques et complets (non symétrique signifie que les poids des arêtes (distances) ne sont pas les mêmes de i à j et de j à i [1])

2 Les méthodes de résolution de ce projet

2.1 Partie 0 : Lecture des données

2.1.1 La bibliothèque TSPLIB

Le site `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/` est entièrement dédié au problème du voyageur de commerce.

Le paragraphe "Symmetric traveling salesman problem (TSP)" contient des jeux de données et, dans de nombreux cas, la solution optimale.

Les instances (jeux de données) sont écrites au format texte dans des fichiers avec une structure décrite dans la FAQ et la documentation (fichier "tsp95.pdf ").

Nous n'allons pas gérer toutes les options des fichiers, seulement les cas suivants :

```
TYPE : TSP c'est à dire le problème symétrique et complet
EDGE_WEIGHT_TYPE : EUCL_2D, GEO, ATT
NODE_COORD_SECTION : entiers ou flottants.
```

Attention : si EUCL_2D utilise la distance euclidienne classique, GEO utilise la distance géographique, et ATT une distance euclidienne modifiée (voir documentation).

La structure de données que vous choisirez pour représenter les instances et les tournées devra intégrer les informations des fichiers lus.

2.1.2 Les packages Python à utiliser

Un problème des programmes d'optimisation est qu'ils peuvent parfaitement fonctionner du point de vue informatique, mais faire des calculs faux et donc rendre des résultats sans intérêt. Encore faut-il le détecter.

Il existe un package Python : tsplib95, (`https://tsplib95.readthedocs.io/en/stable/pages/usage.html`) qui implémente la lecture des fichiers de TSPLIB, et permet des faire divers calculs. Nous allons utiliser ce package pour faire des vérifications des calculs du programme C et faire des tests automatiques avec un autre package : "subprocess".

Le package Python "subprocess" permet de lancer une commande externe comme un programme C, d'interagir avec ce dernier comme lui fournir des données, et de récupérer le résultat. Le programme *test_tsp_c.py* donne un exemple d'interaction Python-C.

1. Par exemple une ville comporte des voies à sens unique.

2.1.3 Factorisation du code et généricité

Le fonction calculant la longueur totale d'une tournée doit pouvoir utiliser différentes fonctions de distance. Il convient donc de la déclarer avec un pointeur de fonction permettant de lui communiquer la fonction de distance correspondant à l'instance lue.

De plus, dans certains cas, on peut programmer dans un esprit de généricité. C'est le cas pour la recherche en force brute et pour les algorithmes génétiques. Le code C devra s'abstraire le plus possible du cas particulier du problème du voyageur de commerce, dans une optique de réutilisation du code pour un problème de nature voisine. La demande sera précisée dans les paragraphes décrivant les algorithmes concernés.

2.1.4 Implémentation

- Implémentez en C des structures de données qui sera commune à tout le projet pour représenter les instances et les tournées
- Implémenter en C une fonction de lecture des fichiers TSPLIB
- Implémenter en C les fonctions de distance et de longueur d'une tournée avec un pointeur de fonction pour la distance.
- Faire une fonction de calcul de la demie-matrice inférieure des distances à partir des coordonnées des points. Cette fonction recevra la fonction de distance qui convient pour ce calcul. La matrice sera dynamique et optimisera l'espace mémoire. Comme le problème est symétrique, on n'a besoin de représenter que la moitié des cases. On utilisera donc un tableau dynamique bi-dimentionnel ayant exactement le bon nombre de cases. En clair chaque ligne a une case de moins que la précédente. Pour avoir la distance entre i et j on accède à $M[i][j]$ si $j > i$ et à $M[j][i]$ sinon.
- Faire un main C, admettant en paramètre de la ligne de commande, la balise -f suivie d'un nom de fichier et -c, affichant les données lues et calculant la longueur de la tournée canonique [2].
- Vérifiez le bon fonctionnement du programme Python *test_ tsp_ c.py* concernant la lecture des fichiers de TSPLIB, le calcul des longueurs de tournées, l'appel du programme C avec l'option -c et la cohérence des résultats.
- Vérifiez vos fonctions de distance en comparant la longueur des tournées canoniques calculées par votre programme avec ce qui est calculé par la fonction de tsplib95 (voir documentation).

2.2 Partie 1 : L'algorithme de force brute (aka brute force)

Puisqu'il s'agit de trouver l'ordre de visite des villes minimisant le trajet total, une idée serait de générer toutes les possibilités et de retenir celles (oui il peut y en avoir plusieurs) qui donne la distance minimale. Comme on ne saura qu'à la fin laquelle est la meilleure, on peut tant qu'on y est, mémoriser la pire pour avoir un point de comparaison.

Le nombre de solution est le nombre de permutations de l'ensemble des villes, soit $N!$ [3] pour N villes. Par exemple pour 27 villes on a un nombre de permutations à tester de :
10888869450418352160768000000.

En clair s'il y a N sommets dans le graphe, il faut calculer les longueurs des $(N-1)!$ tournées possibles ($N-1$ car on part toujours du même point). On trouve à coup sûr l'optimum mais l'algorithme est inefficace.

A moins d'avoir accès à un super calculateur du top ten mondial [4], un tel algorithme demande des heures, voire des jours, mois et même années pour trouver la solution optimale. Il n'est donc pas exploitable pour l'industrie électronique, néanmoins il peut être utile pour trouver la solution optimale d'une instance de petite

2. La tournée canonique est la tournée $[1, 2, 3, ..., N]$ où N est le numéro du dernier point.
3. En réalité, toutes les permutations circulaire d'une permutation vont donner le même résultat. Générer l'ensemble des permutations débarrassé des permutations circulaires est un problème compliqué.
4. http ://www.top500.org/

taille (de l'ordre de 10 à 20 villes) pour mettre au point le code C implémentant les méthodes présentées ci-après.

Exemple :

Parmi les jeux de données fournis sur Moodle figurent deux fichiers *att10.tsp* et *att15.tsp* formés respectivement des 10 et 15 premiers noeuds du fichier *att48.tsp* de TSPLIB. ATTENTION : ces instances utilisent une distance particulière qui n'est pas la distance euclidienne classique (voir plus loin).

Pour éviter de saturer la mémoire de la machine, nous utiliserons la méthode qui, à partir d'une permutation quelconque, génère la permutation suivante dans l'ordre lexicographique (celui du dictionnaire) et dont nous calculerons la longueur avant de continuer .

L'algorithme de génération de la permutation suivant une permutation donnée, dans l'ordre lexicographique est quelque chose de classique. Voici une référence parmi d'autre :

`https://www.nayuki.io/page/next-lexicographical-permutation-algorithm`

Il suffit donc, à chaque étape, de calculer la permutation suivante dans l'ordre lexicographique et de calculer la longueur de la tournée correspondante qu'on comparera à la meilleure trouvée jusque là. On gardera la plus courte et la plus longue tournée.

Nous allons agrémenter cette partie d'une gestion des interruptions. Un control C normalement, arrête le programme en exécution et rend la main au système. On peut intercepter l'interruption et faire un certain nombre d'actions.

Le programme de force brute devra gérer les interruptions, afficher la meilleure tournée calculée, et demander si on s'arrête ou si on reprend les calculs (voir exemple sur Moodle).

Le programme devra être le plus générique possible. Sa structure est la suivante :

initialisation ;
calculer la première permutation;
repeat
- évaluer la permutation courante;
- mettre à jour la variable gardant la meilleure permutation;

until *il reste des permutations à tester*;
renvoyer la meilleure permutation;

Seule la fonction d'évaluation, ici la fonction de calcul de la distance globale, est dépendante du problème traité. On pourra donc la passer comme pointeur de fonction.

2.2.1 Implémentation

- Implémentez en C, au moyen d'une fonction ou d'une procédure, l'algorithme générique de force brute trouvant la tournée optimale et sa longueur.
- Implémenter un main C pour lire un fichier TSPLIB, sélectionner la fonction de distance adéquate (GEO, ATT, EUCL_2D, coordonnées ou matrice), lancer la fonction précédente, et afficher le résultat normalisé (voir annexe).
- Implémenter en C la récupération des interruptions et l'affichage, en cas de control C, de la meilleure tournée au moment de l'arrêt, sa longueur, la permutation courante, et si l'on veut arrêter ou reprendre les calculs.
- Faire des tests sur de petites instances : att5.tsp, att10.tst
- Tester si le fait de calculer la matrice des distances au départ, et de l'utiliser ensuite, est rentable par rapport au calcul des distances à la demande.

— Utilisez le programme Python *test.py* pour lancer le programme C et vérifier le résultat (sur le petites instances).

2.3 Partie 2 : Les algorithmes utilisant des heuristiques

Définition complètement informelle : Une heuristique est une méthode ad hoc permettant de guider un calcul dans une résolution de problème.

2.3.1 Heuristique du plus proche voisin (nearest neighbour)

Il s'agit de choisir un sommet de départ et de visiter le plus proche voisin de ce sommet et de recommencer de proche en proche. Si on a N sommets il faut N^2 calculs de distance pour trouver une solution. En effet à chaque nouveau point, il faut calculer les distances avec les, au plus, $n-1$ autres points pour choisir le suivant. L'algorithme est efficace mais la solution n'est pas optimale en général.

2.3.2 Heuristique de la marche aléatoire (random walk)

On choisit à chaque étape un sommet de manière aléatoire parmi ceux qui restent à visiter. Là encore on fait environ N choix pour obtenir une tournée qui ne sera pas optimale.

2.4 La 2-optimisation : améliorer une solution existante

Théorème : si deux arêtes d'une tournée se croisent alors les décroiser diminue la longueur totale.

La 2-optimisation (ou 2-opt) est une méthode pour améliorer une tournée. Elle consiste à éliminer deux trajets (arêtes) non consécutifs dans la tournée et reconnecter la tournée d'une manière différente. Cela sera intéressant si les deux arêtes se croisaient au départ.

On peut appliquer systématiquement ce principe en recherchant pour chaque arête de la tournée, s'il y a une ou plusieurs autres arêtes qui la croissent et faire une 2-opt.

Il faut donc initialiser le principe avec une tournée trouvée par une des méthodes précédentes et appliquer ensuite la 2-opt. On améliore la tournée, le nombre de calculs reste de l'ordre de N^2, et la tournée n'est toujours pas optimale. Elle peut cependant être raisonnablement bonne.

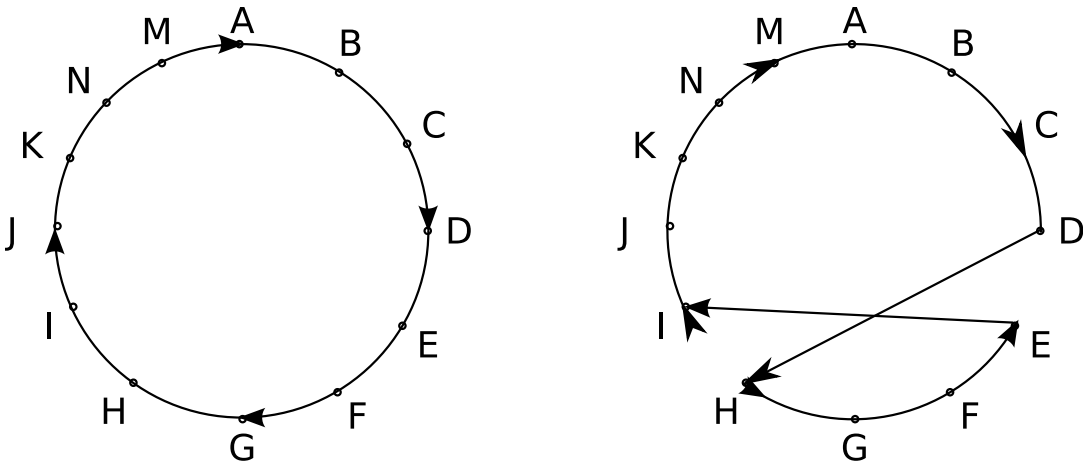

FIGURE 1 – Supprimons les trajets DE et HI. On reconnecte D avec H et E avec I.

2.4.1 Implémentation de ces méthodes

Il s'agit de suivre les explications données ci-dessus.
— Implémentez en C l'algorithme du plus proche voisin (fonction ou procédure)
— Implémenter en C l'algorithme de la marche aléatoire (fonction ou procédure)
— Implémenter en C le 2-opt avec une tournée calculée par une des deux méthodes précédentes (fonction ou procédure)
— Faire un main avec comme paramètre de la ligne de commande -f suivi du nom du fichier, -m suivi de nn, rw, 2optnn ou 2optrw
— Modifiez le programme Python *test_ tsp_ c.py* pour vérifier les calculs.

2.5 Partie 3 : Algorithmes évolutionnaires : algorithme génétique générique

Les algorithmes génétiques s'inspirent de la biologie (telle qu'elle est comprise par les informaticiens) et calculent des solutions approchées considérées comme "bonnes".

2.5.1 Principe général de la méthode

- créer une population initiale de N individus (tournées) initialisés au hasard (random walk);
repeat
  while *le nombre de croisement voulus n'est pas atteint* **do**
    - sélectionner au hasard deux individus;
    - faire des croisement eux qui donnent une tournée fille;
    - avec une probabilité p faire muter la fille;
    - remplacer un individu de la population par la fille (au hasard ou le moins performant);
  end
until *le nombre fixé de générations ou la stabilité est atteint*;

Paramètres possibles ici :

— Nombre d'individus : 20, 50, 100, ...
— Nombre de croisements par génération : de 1 à la moitié de la population ou plus.
— Taux de mutation : 0.1, ...
— Nombre de générations : 200, 1000, 5000, ...

L'utilité de la mutation est d'introduire des perturbations dans ce principe pour maintenir une diversité, et réduire le risque d'être piégé dans un optimum local.

L'algorithme présenté ci-dessus peut avoir de très nombreuses variantes.

2.5.2 Tests de trois variantes avec Python

Les programmes Python *tsp.py*, *tsp-tri.py* et *tsp-tri-light.py* implémentent des variantes d'un algorithme génétique.

2.5.3 Travail de recherche et implémentation en C

— Pour les trois programmes Python, faîtes varier les paramètres et sélectionnez le programme qui vous semble le plus performant
— Implémentez en C l'algorithme Python que vous avez sélectionné. Restez génériques dans le codage. Les balises du programme sont présentées en annexe.

2.5.4 Variante *ad hoc* : Le croisement DPX (distance preserving crossover)

Nous introduisons ici une variante ad hoc qui rompt avec la généricité en remplaçant le croisement standard par un croisement spécifique au TSP : le DPX.

Etant donné deux tournées dites parent-1 et parent-2, on initialise la tournée fille en copiant le parent-1. Puis toutes les arêtes qui ne sont pas en commun avec le parent-2 sont détruites. Les morceaux déconnectés de tournée sont recombinées par la méthode suivante :

si le trajet (i, j) a été détruit, alors si k est le plus proche voisin de i parmi les extrémités des autres fragments, on ajoute le trajet (i, k) (si ce trajet n'est contenu dans aucun des deux parents) et tous les trajets du fragment de k en le renversant si nécessaire.

Exemple :
Avec $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ l'ensemble des villes :

Parent 1 = $(5, 3, 9, 1, 2, 8, 0, 6, 7, 4)$
Parent 2 = $(1, 2, 5, 3, 9, 4, 8, 6, 0, 7)$

Fille = $(5, 3, 9)(1, 2)(8)(0, 6)(7)(4)$

Trajets détruits : $(9, 1), (2, 8), (8, 0), (6, 7), (7, 4)$

Après reconnexion, Fille = $(5, 3, 9, 8, 7, 2, 1, 4, 6, 0)$

Avec ppv(9)=8, ppv(8)=7, ppv(7)=2, ppv(1)=4, ppv(4)=6 et ppv = plus proche voisin.

1. Utilisez le DPX à la place du croisement précédent et tester son efficacité.
2. Ajoutez une 2-opt de la tournée fille

3 Annexe : informations diverses et spécifications des formats d'entrée/sortie

Documentation du package TSPLIB :

`https://tsplib95.readthedocs.io/`

3.1 Les options de la ligne de commande des programme C

Pour contrôler les programmes des différentes parties, on utilisera les balises suivantes :

```
-h : help, affiche l'usage et ne fait aucun calcul.
-f <nom de fichier TSPLIB>
[-o <nom de fichier de sortie>] balise optionnelle pour sauver dans un fichier en mode append.
-m <méthode de calcul>  parmi :
```

- bf : force brute
- nn : plus proche voisin (nearest neibourgh),
- rw : marche aléatoire (random walk),
- 2optnn : 2-opt avec initialisation par le plus proche voisin
- 2optrw : 2-opt avec initialisation par la marche aléatoire,
- ga <nombre d'individus> <nombre de générations> <taux de mutation> : algorithme génétique générique,

— gadpx suivi des mêmes paramètres que ga mais utilisant le DPX et la 2opt.
— all : toute les méthodes sauf la force brute. Dans ce cas l'entête des résultats ne sera affichée qu'une fois.

Remarque :
— Pour gérer les balises des programmes main en C, le header getopt.h peut vous être utile.
— Prévoir un contrôle des erreurs possibles dans les balises et les paramètres.

3.2 Normalisation des affichages

Le programme Python $test_tsp_c.pyattend, delapartduC, desaffichageshomognespourpouvoirfonctionner.Voiciunexemple$

```
Instance ; Méthode ; Temps CPU (sec) ; Longueur ; Tour
burma14 ; nn ;   0.00 ;    4048.00 ; [1,8,11,9,10,2,14,3,4,12,6,7,13,5]
att48 ; rw ;   0.00 ;   46020.00 ; [1,33,15,22,3,29,8,45,28,23,11,25,37,10,32,39,4,6,46,18,34,47,43,36,
44,12,35,26,16,17,19,20,31,30,21,48,38,2,7,9,5,27,40,24,42,41,13,14]
```

La bannière (première ligne) n'est affichée qu'une fois avec la balise all. *Le caractère ";" permet, si les résultats sont sauvés dans un fichier suffixé "csv", de le charger dans un tableur. Vous remarquerez que le programme Python reprend le même principe d'affichage :*

```
Instance ; algo ; long (C) ; long (Python) ; temps ; tour ; valid ; mêmes longueurs
att48 ; canonical ; 49840 ; 49840 ; 0.0 ; [1, 2, 3, 4, 5, ..., 43, 44, 45, 46, 47, 48] ; True ; True
att48 ; nn ; 12861 ; 12861 ; 0.0 ; [1, 9, 38, 31, 44, 18, 7,..., 42, 26, 4, 35, 45, 2, 8] ; True ; True
att48 ; rw ; 50672 ; 50672 ; 0.0 ; [1, 7, 30, 40, 23, 5, ...,,4, 8, 34, 31, 13] ; True ; True
att48 ; 2optnn ; 12272 ; 14166 ; 0.01 ; [1, 2, 42, 26, 4, ...,  16, 41, 34, 29, 5] ; False ; False
```

On confronte les calculs du C avec ceux de Python qui sont sensés être exacts (affichage direct et booléen "mêmes longueur"). On vérifie aussi la cohérence de la tournée calculée (valid).

Remarque *: Dans l'exemple ci-dessus, les listes des tournées ont été tronquées pour clarifier l'affichage dans le pdf.*