Résolution pratique du TSP

Objectif: L'objectif du TP est d'illustrer la notion d'heuristiques et la mise en oeuvre et l'expérimentation de quelques méthodes -approchées ou exactes- pour résoudre un problème NP-dur.

Le Problème

Le problème que nous considérons correspond au problème de décision du voyageur de commerce étudié au TP précédent. Dans sa version optimisation, le problème du voyageur de commerce consiste, étant donné un ensemble de n villes séparées par des distances données, à trouver le plus court circuit qui relie toutes les villes.

Travelling Salesman Problem (TSPOpt)

Donnée

n, un entier – un nombre de villes

D, une matrice (n, n) d'entiers – elle représente les distances, qu'on suppose entières

Sortie

Une tournée de longueur minimale, i.e.

$tour : [0 \dots n-1] \to [0 \dots n-1]$ – une **permutation** des n villes

telle que:

$D(tour[n-1], tour[0]) + \sum_{i=0}^{n-2} D(tour[i], tour[i+1])$ soit minimale.

Exemple: soit $n = 4$ et D donnée par les distances suivantes:

	A	B	C	D
A	0	2	5	7
B	7	0	8	1
C	2	1	0	9
D	2	2	8	0

La longueur minimale d'une tournée est 9 avec la tournée représentée par la permutation A, C, B, D.

Remarque importante: dans la spécification du problème, les distances entre les villes sont entières comme dans le précédent TP. Plusieurs jeux de données de ce type sont à votre disposition sur le portail. Cependant, dans beaucoup de benchmarks, les données sont représentées par les coordonnées des villes et donc ensuite la matrice calculée des distances est à valeurs réelles. Vous pouvez vous placer dans ce cas d'emblée si vous souhaitez pouvoir traiter de tels jeux de données.

À faire pour la semaine du 10 décembre

Nous vous proposons plusieurs pistes, mais d'une part, il n'est absolument pas requis de toutes les implémenter, d'autre part, vous pouvez aussi choisir d'en implémenter d'autres. En particulier, **il est facultatif d'implémenter une méthode exacte** -si vous le faîtes, cela pourra être considéré comme un Bonus dans votre notre de contrôle continu. Donc, il est demandé de:

- Implémenter plusieurs méthodes, au minimum deux méthodes approchées, une "globale" et une méthode basée sur la notion de voisinage.
- Expérimenter sur les jeux de tests donnés ou d'autres benchmarks disponibles (voir par exemple). Attention, aux différentes variantes du problèmes - par exemple symétrique ou non- et à la syntaxe des jeux de données.
- Produire un court compte-rendu expliquant vos choix (quelles méthodes, quelle implémentation, ...), vos résultats d'expérimentation, et éventuellement les réponses aux quelques questions.

1 Heuristiques globales

De nombreuses heuristiques ont été proposées pour le problème. Nous vous en proposons deux ci-dessous, mais vous pouvez choisir d'en implémenter une autre si vous le souhaitez.

Une solution pour le TSP est une permutation des villes. Les deux heuristiques suivantes construisent incrémentalement cette permutation de manière différente.

1.1 Construction itérative par ajout du plus proche

La première heuristique construit cette permutation ville par ville en partant du début. Elle peut s'écrire de la manière suivante :

1. Initialiser une permutation avec le premier sommet.
2. A chaque étape choisir la plus proche ville du dernier sommet visité avec une ville non encore sélectionnée et l'ajouter au tour.
3. S'arrêter lorsque l'on a sélectionné toutes les villes.

Dans cette méthode, on a à toute étape un chemin de longueur inférieure ou égale à n reliant la première ville à une autre. L'algorithme s'arrête lorsque le chemin est de taille $n-1$. Il ne faut pas oublier d'ajouter le retour à la ville de départ.

1.2 Construction par ajout d'arcs

Une autre heuristique construit le circuit en ajoutant des arcs itérativement. Dans cette version, une solution partielle n'est pas un début de tournée mais plutôt un ensemble de tronçons de tournée.

1. Trier les arêtes par valeur croissante.
2. A chaque étape, ajouter au tour l'arête courante si elle ne crée par de circuit de taille inférieure à n.

On a donc à toute étape une liste d'arcs sélectionnés, non obligatoirement consécutifs. Il faut s'assurer à chaque étape que : deux arcs sélectionnés ne partent du même sommet, que deux arcs sélectionnés n'arrivent pas au même sommet, et qu'on ne crée pas de sous-tour (circuit de taille au plus $n-1$).

Q 1. [A coder] Implémenter au moins une heuristique.

Q 2. Expérimenter sur le jeu de tests donné.

Q 3. Si on suppose que la distance vérifie l'inégalité triangulaire, il existe des heuristiques polynomiales avec garantie. Montrer que si ce n'est pas le cas (i.e. la distance vérifie l'inégalité triangulaire), il ne peut exister d'heuristique polynomiale avec garantie..

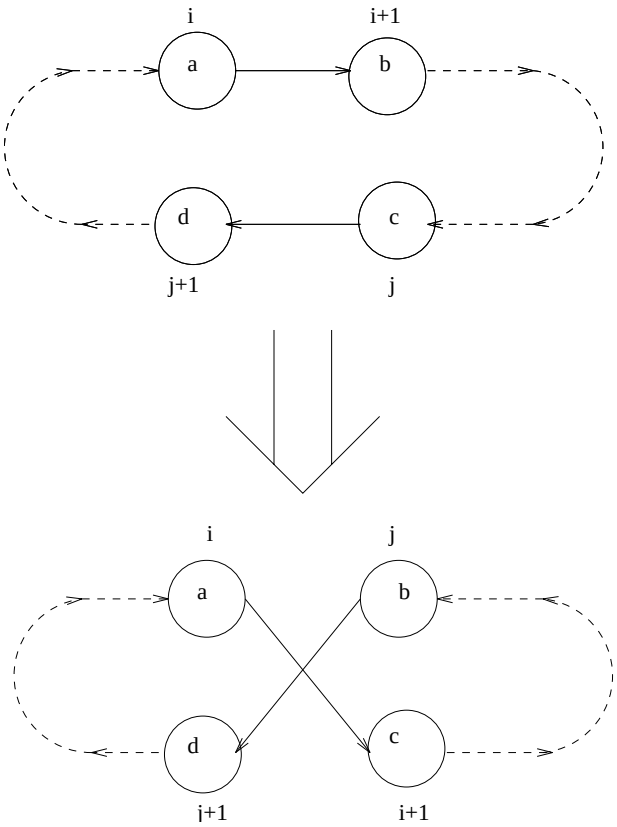
Aide: Utiliser le fait que l'existence d'un cycle Hamiltonien soit $NP-dur$.

2 Méthodes locales

Nous allons travailler sur des méthodes locales : on part d'une tournée complète et à chaque étape on la transforme en une tournée "voisine". Les trois méthodes que nous allons envisager sont basées sur la même notion de voisinage. Là encore, vous pouvez aussi choisir d'autres méthodes... Vous pouvez aussi si vous le préférez choisir d'implémenter d'autres types de méthodes, comme un algorithme par colonies de fourmis.

On pourra supposer ici que la matrice de distance est symétrique, i.e. que D(i,j)=D(j,i) pour tout couple (i,j).

La notion de voisin: On doit donc d'abord définir la notion de tournée "voisine"; une tournée T' est dite voisine de la tournée T si on l'a obtenue à partir de T par la transformation représentée comme suit:



Soit formellement, à une tournée T, et à un couple (i, j) d'entiers tel que $1 < i < j - 1 < n$ ou $i = 1 < j - 1 < n - 1$, on associe une tournée voisine T' définie par:

$$T'(k) = T(k), k <= i.$$
$$T'(i + 1) = T(j)$$
$$T'(j + 1 - p) = T(i + p), 1 \leq p \leq j - i$$
$$T'(k) = T(k), k > j$$

Donc pour une tournée donnée, une tournée voisine est donnée par un couple (i, j) d'entiers tel que $1 < i < j - 1 < n$ ou $i = 1 < j - 1 < n - 1$.

2.1 Recherche d'un optimum local ou "HillClimbing"

On part d'une solution si possible "bonne" (par exemple donnée par une heuristique ci-dessus) et on balaie l'ensemble des tournées voisines de cette solution; si il n'existe pas de voisin meilleur que notre solution, notre solution est un optimum local et on arrête; sinon, on choisit le meilleur des voisins et on recommence -une variante consiste à passer au premier voisin rencontré meilleur que la solution actuelle. On peut éventuellement se fixer un nombre de boucles maximum, si on veut limiter le temps d'exécution.

Remarque: il n'est pas nécessaire de construire toutes les tournées voisines: on évalue leur coût et on ne construit que la meilleure.

Cette méthode a l'inconvénient de "rester bloquée" dans un optimum local: une fois un optimum local trouvé, on s'arrête, même si ce n'est pas l'optimum global. Les deux méthodes qui suivent, permettent de poursuivre la recherche même après avoir obtenu un optimum local.

2.2 La méthode "Tabou"

A chaque étape, on recherche le meilleur voisin, mais on limite la recherche aux voisins non tabous: un voisin est à priori tabou si on a exploré cette solution durant les N précédentes itérations. Mais la maintenance d'une liste taboue de tournées est trop coûteuse; on se borne donc à éviter de faire la transformation inverse d'une transformation effectuée durant les N précédentes itérations; on va donc stocker les transformations effectuées, c.a.d. uniquement les couples d'arêtes correspondant. A chaque itération, on choisit le meilleur voisin correspondant à une transformation non taboue; on effectue cette transformation, on la place dans la

liste (file) taboue et on élimine la plus ancienne transformation de cette liste. De plus, si la tournée actuelle est la meilleure trouvée depuis le début, on la stocke (la meilleure tournée ne sera pas forcément obtenue à la fin !). On s'arrête soit après un nombre fixé d'itérations, soit après un nombre fixé d'étapes n'ayant pas amélioré la solution.

Cette solution pourra donc être paramétrée par la longueur de la liste taboue: vous pouvez faire varier cette longueur entre 10 et 20.

2.3 Le recuit simulé

A chaque étape, on tire au hasard une tournée voisine de la tournée actuelle. Si elle est meilleure, on l'adopte; sinon on calcule l'augmentation de coût Δ; on va adopter cette solution avec une probabilité liée à cette augmentation: plus le coût augmente, plus la probabilité de la garder sera faible.

Plus précisément on calcule $accept = E^{-\Delta/T}$, T étant une quantité que nous appelerons *la température* (nous y reviendrons). On tire au hasard un réél p entre 0 et 1: si p est inférieur à $accept$, on adopte la nouvelle tournée, sinon on garde l'ancienne.

La température et le refroidissement: il faut donc choisir une température initiale, et pour assurer l'arrêt, une façon de diminuer la température au cours de l'exécution.

Le choix de la température initiale: La température initiale doit être assez élevée pour que $accept$ soit assez grand au départ même si l'augmentation de coût est grande. On pourra par exemple choisir T_0 pour que $E^{-\Delta_0/T_0} = 2/3$, avec $\Delta_0 = 2 * max(d(v_i, v_j))$. La température initiale devrait être paramétrable, par exemple donnée par l'utilisateur.

Le refroidissement: Pour diminuer T, on peut par exemple multiplier T par un réél inférieur à 1 à chaque fois. Plus ce réél est proche de 1, plus le refroidissement est lent. Ce "coefficient de refroidissement" est lui aussi un paramètre fixé lors de l'utilisation.

On peut aussi raffiner: on effectue plusieurs cycles de recuit simulé, en repartant à chaque fois de la solution trouvée au cycle précédent.

À faire:

Q 1. Programmer une ou deux méthodes! Vous pouvez vous partager le travail. Il peut être intéressant de prévoir un mode trace pour suivre l'évolution d'une tournée au cours de l'exécution de ces méthodes "locales".
Q 2. Comparer les méthodes sur les jeux de test donnés ou sur des benchmarks classiques.

3 Méthodes exactes

On a proposé dans le montré dans le TP précédent un algorithme exact pour répondre au problème de décision. On peut bien sûr l'adapter pour répondre au problème d'optimisation qui nous intéresse ici. Cet algorithme sera bien sûr exponentiel et très vite impraticable même sur des données de taille assez petite.

On peut aussi essayer de raffiner cette recherche exhaustive en utilisant des méthodes de type "Branch and Bound" ou "séparation, évaluation" qui évitent de parcourir l'espace de toutes les solutions en "élaguant" (pruning), par exemple en évitant de développer une solution partielle si on est sûr qu'elle sera moins bonne qu'une solution déjà trouvée. Ce type de méthode Branch and Bound ou Séparation-Evaluation qu'on peut voir comme une recherche dans l'arbre des solutiuns est basé sur le schéma général suivant:

- Séparation (Branch): à partir d'un noeud, on explore les différentes branches
- Evaluation (Bound): on évalue l'intérêt des branches.
  - On essaie de trouver une borne inférieure: toute solution issue de ce nœud aura un coût au moins de MIN, voire une borne sup: toute solution issue de ce noeud aura un coût au plus de MAX.
  - Si la borne inférieure d'un nœud A est supérieure au coût de la meilleure solution déjà trouvée, il est inutile de l'explorer: c'est l'*élagage* ou le *pruning* (de même si elle est supérieure à la borne supérieure de B).
  - On peut explorer en premier les branches les plus prometteuses.

Ici un nœud correspondra donc à une solution partielle. Comme dans la première partie, une solution partielle peut-être un début de tournée, ou un ensemble d'arcs qui pourront construire une tournée.

3.1 Une première approche très simple

Une première approche très simple consiste donc à considèrer une tournée partielle comme un début de tournée. Les différentes branches à partir d'un nœud correspondent donc aux différents choix de la prochaine ville. Une borne inférieure pour toute solution issue de ce nœud peut être simplement la longueur du début de tournée à laquelle on ajoute $(x + 1) * dmin$, si x est le nombre de villes qu'il reste à visiter, $dmin$ le minimum de la distance entre deux villes différentes. dans cette méthode.

Q 1. Programmer et expérimenter cette approche.

3.2 Une approche un peu plus sophistiquée

3.2.1 Borne inférieure d'une tournée

On peut raffiner un peu la calcul de la borne inférieure d'une tournée. Calculer une telle valeur a plusieurs intérêts.

1. Dans le cas du problème de décision, il possible de répondre "non" facilement dans certains cas.
2. Cela permet d'estimer la qualité des heuristiques en les comparant à la valeur obtenue.
3. Cela permet d'accélérer des méthodes exactes.

3.3 Description de la borne

Une première borne inférieure peut être obtenue de la manière suivante. Si la distance minimale entre deux villes est k, on ne trouvera jamais de tournée de valeur inférieure à k. En sommant pour toutes les villes, on obtient une borne égale à nk.

On peut généraliser le raisonnement de la manière suivante. On sait que dans un tour, on va devoir emprunter un chemin sortant de chacune des villes. Par conséquent la somme des distances minimales sortant de chaque ville est une borne inférieure pour la valeur optimale d'un tour.

La méthode suivante est basée sur cette idée.

1. pour chaque ligne de la matrice de distances, soustraire la valeur minimum sur cette ligne
2. pour chaque colonne de la matrice obtenue, soustraire la valeur minimum sur cette colonne

	A	B	C	D
A	∞	5	7	8
B	10	∞	16	10
C	7	5	∞	2
D	10	11	17	∞

	A	B	C	D
A	∞	0	2	3
B	0	∞	6	0
C	5	3	∞	0
D	0	1	7	∞

	A	B	C	D
A	∞	0	0	3
B	0	∞	4	0
C	5	3	∞	0
D	0	1	5	∞

Table 1: Prétraitement sur la matrice de distances : la matrice initiale, la matrice obtenue en soustrayant le minimum sur chaque ligne et la matrice obtenue en soustrayant le minimum sur chaque colonne. La borne est égale à $(5 + 10 + 2 + 10) + 2 = 29$.

On remarque que cette méthode permet à la fois d'obtenir une borne inférieure et de simplifier la donnée (on remarque qu'on obtient au moins un zéro par ligne et par colonne).

Q 2. Montrer que toute solution optimale du problème réduit est solution optimale du problème initial.

Q 3. Comment obtenir la valeur de la solution du problème initial en fonction de la valeur de la solution du problème réduit ?

Q 4. Ecrire une méthode qui calcule une borne inférieure pour le problème de tournée et qui calcule une donnée simplifiée.

Q 5. Quelle est la complexité de cette méthode ?

3.4 Méthode exacte de Little

On propose ici l'algorithme de Little, une méthode de résolution exacte basée sur une recherche arborescente. Pour éviter une énumération complète, on ajoute une méthode qui va évaluer pour chaque nœud de la recherche s'il peut mener à une meilleure solution. Le travail à effectuer est indiqué pour chaque étape.

3.4.1 Version basique de la recherche arborescente

À un nœud donné de l'arbre de recherche, on dispose d'une liste d'arcs sélectionnés et d'une matrice courante de distances.

Un nœud correspond à une solution partielle (une liste d'arcs sélectionnés). A la racine, la liste est vide. Au cours de l'énumération, on va ajouter ou interdire des arcs jusqu'à obtention d'une solution ou épuisement des arcs.

Le schéma de séparation est le suivant : à partir du nœud courant, on sélectionne un arc (i, j) (ni ajouté ni interdit) de valeur différente de $+\infty$ et on crée deux nouveaux nœuds (qu'on appellera "fils" du nœud courant) : l'un où l'on ajoute (i, j), l'autre où l'on interdit d'ajouter (i, j).

Dans un premier temps, on peut choisir le premier couple (i, j) tel que $M_{ij} \neq +\infty$.

Q 6. Si on sélectionne l'arc (i, j), quels arcs peut-on interdire automatiquement ? On pensera notamment à l'arc (j, i).
Q 7. Comment évolue le coût minimum de la solution courante lorsque l'on ajoute un arc ?
Q 8. Comment modifier le problème dans le deuxième fils pour s'assurer que l'arc (i, j) ne sera pas choisi ?

On est arrivé à une feuille lorsqu'on a obtenu n arcs dans la solution partielle. Une feuille correspond à une solution réalisable s'il n'y a pas de sous-tour.
Q 9. Implémenter la recherche arborescente basique.

3.4.2 Introduction de la borne inférieure

On peut améliorer la méthode en stoppant l'évaluation d'un nœud de l'arbre lorsque la valeur de sa solution partielle est supérieure à la meilleure solution déjà trouvée.
Q 10. Lorsqu'un arc est interdit, comment mettre à jour la borne de la section 3.2.1?
Q 11. Même question lorsqu'un arc est ajouté. Ne pas oublier que l'ajout d'un arc provoque l'interdiction d'autres arcs.
Q 12. Intégrer cette borne dans votre algorithme.

3.4.3 Choix de la séparation

Le choix du couple (i, j) sur lequel la séparation s'effectue est important pour l'efficacité de la méthode.

Si l'on veut arriver rapidement à une bonne solution, une première idée est de choisir toujours un arc dont la valeur est zéro (il en existe obligatoirement si on a utilisé la technique de borne inférieure décrite plus haut).

Parmi les arcs de valeur 0, on choisira celui qui occasionnerait le plus grand surcoût s'il n'était pas choisi. Pour ce faire, pour chaque arc (i, j) de coût 0, on calcule la valeur minimum d'un arc $(v, j), v \neq i$ et $(i, v), v \neq j$.
Q 13. Introduire le choix du meilleur arc à insérer dans la méthode.