TECHNIQUES ALGORITHMIQUES ET PROGRAMMATION

Voyageur de commerce

On rappelle la définition du problème :

> VOYAGEUR DE COMMERCE :
>
> **Instance :** Un ensemble $V = \{v_0, \ldots, v_{n-1}\}$ de points et une distance d sur V.
>
> **Question :** Trouver une tournée de longueur minimum passant par tous les points de V, c'est-à-dire une permutation σ des indices des éléments de V telle que $\sum_{i=0}^{n-1} d(v_{\sigma(i)}, v_{\sigma(i+1 \bmod n)})$ est minimum.

Dans la suite on supposera que $V \subset \mathbb{R}^2$ est un ensemble de n points du plan et que d est la distance euclidienne entre deux points. L'objectif des TP va être de programmer et de tester les performances de plusieurs algorithmes sur certaines instances du problème.

On utilisera la structure de donnée `point` défini par `struct{ double x,y; }`.

Question 1. *Donner le code* `C` *de la fonction* `dist(A,B)` *donnant la distance entre les points* `A` *et* `B`.

On représentera une permutation σ de $\{0, \ldots, n-1\}$ par un tableau `P` de taille n tel que `P[i]` $= \sigma(i)$. Par exemple, `int P[]={3,1,2,0};` représentera une permutation pour $n = 4$ qui inverse le premier avec le dernier élément, ce qui définit la tournée v_3, v_1, v_2, v_0, v_3. Dit autrement, `P` représente l'ordre de visite des points de V dans la tournée.

Question 2. *Écrivez la fonction* `value(point *V,int n,int *P)` *qui renvoie la valeur de la tournée des* `n` *points de* `V` *selon la permutation* `P`.

1 Approche « Brute-Force »

Les permutations peuvent être rangées par ordre lexicographique de la plus petite (dans notre exemple `P={0,1,2,3}`) à la plus grande (`P={3,2,1,0}`). On suppose donnée la fonction `int NextPermutation(int *P,int n)` qui calcule, en mettant à jour `P`, la permutation immédiatement suivante dans l'ordre lexicographique. De plus, la fonction renvoie faux ($= 0$) si et seulement si, avant l'appel à la fonction, la permutation `P` correspondait à la plus grande permutation. Vous pourrez utiliser, mais c'est pas nécessaire, la constante `DBL_MAX` qui définit le `double` de plus grand (définie dans `<float.h>`).

Question 3. *Écrivez la fonction* `double tsp_brute_force(point *V,int n,int *Q)` *qui à partir d'un ensemble* `V` *de* `n` *points renvoie la permutation* `Q` *minimisant la valeur de la tournée selon l'approche exhaustive (ou « Brute-Force ») ainsi que la valeur de cette tournée.*

On va maintenant optimiser la procédure précédante en `tsp_brute_force_opt()`. La première optimisation consiste à fixer l'un des points de la permutation, le premier ou le dernier (à voir ce qui est le plus pratique). Cela permet de gagner un facteur n sur le nombre de permutations à tester.

La seconde consiste, pendant le calcul de `value()`, d'arrêter l'évaluation dès que la distance courante dépasse la meilleure tournée déjà obtenue, disons w. Dans cette optimisation, n'oubliez pas le retour au point de départ ! c'est-à-dire qu'une tournée partielle utilisant les $i+1$ premiers points $v_{\sigma(0)}, \ldots, v_{\sigma(i)}$ aura pour valeur $w_i = \left(\sum_{j=0}^{i-1} d(v_{\sigma(j)}, v_{\sigma(j+1)})\right) + d(v_{\sigma(i)}, v_0)$. L'observation est que si $w_i \geqslant w$, alors on peut directement passer à la plus grande permutation de préfixe $\sigma(0), \ldots, \sigma(i)$.

Question 4. *Écrivez une fonction* `double value_opt(V,n,P,w)` *qui renvoie :*
- *la valeur de la tournée si elle est plus petite que* `w` *; ou bien*
- `-k` *si* `k` *est le nombre d'arêtes de la première tournée partielle qui dépasse* `w` *(faire attention à l'indice renvoyé).*

Question 5. *Écrivez une fonction* `MaxPermutation(P,n,k)` *qui renvoie dans* `P` *la plus grande permutation, dans l'ordre lexicographique, dont le préfixe de taille* `k` *est celui de* `P`*. Quelle la complexité de votre fonction ?*

2 En TP

Télécharger les fichiers correspondant au TP à partir de la page de l'UE disponible ci-après :

`http://dept-info.labri.fr/~gavoille/UE-TAP.html`

Éditer, compiler (`make tsp`) et exécuter le code du fichier `tsp.c`. En particulier, implémenter et tester `tsp_brute_force()` puis que la version optimisée `tsp_brute_force_opt()`.

Commencez par un petit nombre de points ($n = 10$) puis tester les diverses optimisations. Pensez à utiliser la même graine dans le générateur aléatoire, avec `srandom()`, de façon à pouvoir comprarer vos expériences. Vous devriez observer un gain d'un facteur environ 20 pour la version optimisée.