Rapport projet algorithmique :
« Pr oblème du voyageur de commerce »

ARNAUD Alexia
BRAZOUSKAYA Darya
GUNTZ Thomas

Floyd-Warshall :

$d^{(k+1)}(i,j) = min(d^k(i,j), d^k(i,k+1) + d^k(k+1,j))$, cette formule peut s'expliquer de la façon suivante : i et j sont deux sommets du graphe G qui en a n. On cherche le plus court chemin de i à j dont les sommets intermédiaires sont dans 1..k+1.
On se rend compte alors qu'il y a deux possibilités : soit le chemin ne passe pas par le sommet k +1; soit le chemin passe exactement une fois par le sommet k+1 (car il n'y a pas de cycle de coût négatif) et par conséquent le chemin de i à j et la concaténation du chemin de i à k+1 et du chemin de k+1 à j (dont les sommets intermédiaires sont dans 1 ..k pour chacun de ces deux chemins).
La complexité de l'algorithme est $O(n^3)$, en effet il y a 3 boucles for imbriquées : une pour le sommet de départ, une pour le sommet d'arrivée et une pour le sommet intermédiaire.

Énumération :

La complexité pour énumérer toutes les solutions possibles est de O((n-1)!) : on calcule toutes les combinaisons possibles de chemins en partant du premier sommet. On cherche un chemin hamiltonien, donc on ne passe pas deux fois sur le même sommet. Le premier sommet a n-1 arêtes possibles (n-1 sommets non visités), le second sommet aura lui aussi n-1 arêtes moins celle qu'on vient d'utiliser : n-2 et ainsi de suite -> (n-1) !.
Cette complexité croît très vite selon n, pour n>15, le temps de calcul commence à devenir assez long sans pour autant donner une solution pour notre problème. Un algorithme peu viable qui prouve l'intérêt de tester de nouvelles approches.

Algorithme glouton :

La complexité de l'algorithme est $O(n^2)$: on parcourt tous les sommets (au fur et au mesure ils deviennent visités, c'est de l'ordre O(n)) et pour chaque sommet visité on cherche le sommet le plus proche parmi non-visité (ce que est localement optimal, en O(n)), d'où la complexité total. Par contre c'est un algorithme qui trouve pas forcement une solution optimale. On peut traiter les problèmes d'assez grande taille. Cette solution peut être utilisé comme le point de départ pour les autres algo (recherche locale) ou comme la solution réalisable pour évaluation.

Algorithme de recherche locale :

La complexité de l'algorithme est $O(n^2)$ en moyenne (2 boucles for imbriquées). Pour le pire cas le coût peut rapidement devenir exponentiel, cela dépend de la boucle while globale qui a pour paramètre un booléen à vrai tant qu'on peut améliorer la solution.
La taille maximale n du problème que l'on peut traiter est assez grande ; pour n=100, on met 0 .021s.
A partir de quelques échantillons on observe que le pourcentage d'amélioration par rapport à l'algorithme glouton est d'environ : 10 %.
Toutefois, il faut savoir que pour certaines valeurs de n, à certaines exécutions, cet algorithme donne une solution légèrement supérieure à celle de l'algorithme glouton. Malheureusement nous n'avons pas réussi à déterminer la cause de ce petit dysfonctionnement.

Sortir des minimaux locaux:

La complexité de l'algorithme est $O(n^2)$ en moyenne (2 boucles for imbriquées). Pour le pire cas le coût peut rapidement devenir exponentiel, cela dépend de la boucle while globale qui a pour paramètre un booléen à vrai tant qu'on peut améliorer la solution.
La taille maximale n du problème que l'on peut traiter est assez grande ; pour n=100, on met 0 .39s. A partir de quelques échantillons on observe que le pourcentage d'amélioration par rapport à l'algorithme de recherche locale est d'environ: 2% à 3 %.

Programmation dynamique :

L'algorithme de programmation dynamique donne une approche intéressante du problème. La formule énoncée propose de calculer les plus courts chemin pour un ensemble S de sommets. S contient une combinaison de sommets et contient forcément le premier sommet, sa taille grandit à chaque nouvelle itération (appel récursif). Ainsi si 0 est le premier sommet, au premier appel récursif on aura S={0,1}, S={0,2}...S{0,n-1}, le calcul du plus court chemin des S est ici trivial (il correspond à la ligne de transition du sommet 0). Pour le 2nd appel, S={0,1,2}, S={0,1,3}, S={0,2,1}..S{0,n-1,n-2}, le calcul des plus courts chemin se fait grâce au C(k-1) calculé au premier appel + le coût de transition entre les deux derniers sommets (O(n), une seule opération par ensemble). Ainsi de suite jusqu'au n-1 appel récursif, tous les sommets seront dans S et les plus courts chemin seront calculés. Il suffit de prendre le minimum de ces coûts et de rajouter le dernier coût pour revenir au sommet initial.
Complexité : O(n * 2^n) car on dispose de 2^n combinaisons possibles en tout pour S et un simple calcul est fait par combinaison (O(n), on ajoute simplement un coût au résultat précédent déjà calculé).
L'algorithme de programmation dynamique marche très bien pour n<11 (sur les machines Ensimag), il sature à n>12 du fait du trop grand nombre de combinaisons possibles (2^n). Ces combinaisons allouées dynamiquement dans notre programme et arrive rapidement à saturation.

Branch and Bound :

Pour une borne inférieure d'un chemin minimum de type (b; V\S ; a) on a min{distance(b, V\S)} + min{distance(a, V\S)} + poids d'un arbre couvrant minimum de V\S. Car une partie du chemin minimum dans V\S est un chemin hamiltonien, donc un arbre couvrant, donc son poids est supérieur au poids d'un arbre couvrant minimum ; une arête de b à V\S dans le chemin minimum est de poids supérieur au minimum du poids de tous les arêtes de b à V\S (idem pour a). D'où la règle de «bounding».
C'est un algorithme qui trouve la solution optimale, mais dont la complexité augmente très vite avec la taille ($O(n^2 * c^n)$), ou c est une certaine constante, ce terme est du au branchement (mais on ne le fait pas à chaque étape comme on coupe certaines branches), n^2 vient du traitement dans chaque nœud (notamment le calcul de coût de l'arbre couvrant minimum pour construire la borne inférieure). C'est relativement difficile de trouver la taille maximum de l'instance. Sur un ordinateur portable (non Ensimag) on trouve 274 s pour le temps d'exécution avec un graphe complet à 13 sommets.

Algorithme d'approximation :

Complexité :

- tri des arêtes par (Qsort), on a m=n(n-1)/2 arêtes donc le tri se fait en O(mlog(m***)) ;***
- La création de l'arbre couvrant de poids minimum se fait en O(m*n) ;
- La création du tour eulérien se fait en O(m) ;
- Sois u=n-1 le nombre d'arêtes de l'arbre couvrant, la création du tour hamiltonien se fait en O(u) ;
- Donc la complexité de l'algorithme d'approximation équivaut à la somme des complexités ci-dessus ~~ O(m*n).

La solution construite n'est pas forcément optimale, mais elle est au plus 2 fois plus grande que la solution optimale. Cela peut s'expliquer :

- Soit Opt la solution optimale du problème TSP (moins la dernière arête pour obtenir un arbre couvrant) ;
- Soit M l'arbre couvrant minimum ;
- E le tour Eulérien obtenu après duplication des arêtes de M ;
- T la solution approchée ;

On a :
- C(T) <= C(E) obtenu par la propriété d'inégalité triangulaire du graphe (par construction).
- C(E) = 2C(M)
- C(M) <= C(Opt) car M est le minimum de tous les arbres couvrants et Opt est simplement un arbre couvrant.

Donc : C(T) <= 2C(Opt)

Comparaison des différents algorithmes :

Coût moyen calculé pour un n=10 avec 200 itérations.

Algorithmes	Coût en temps en moyennes	Complexité
Énumération	0.626750	O ((n-1)!)
Glouton	0	O (n^2)
Recherche Locale	0	O (n^2)
Sortir des minimums locaux	0	O (n^2)
Programmation dynamique	0,332	O(n * 2^n)
Branch & Bound	0,1186	O($n^2 * c^n$)
Approximation	0.000050	O(m*n)

Graphes générés pour plusieurs valeurs de n, sur 10 itérations (graphe différent par itération).

Les temps d'exécutions utilisés sont la moyenne des temps des 10 itérations.