EISTI Ecole Internationale des Sciences du Traitement de l'Information	**VOYAGEUR DE COMMERCE : ALGORITHME DE LITTLE**
Rédigé par Yannick Le Nir	Ref : CO-*VOY-COM-LIT*
Revu par : l'équipe pédagogique	Crée le 27/10/2012

Sommaire

1 Présentation

Le problème du voyageur de commerce est un incontournable de la recherche opérationnelle. Il appartient à la classe des problèmes complexes en optimisation combinatoire et sa proximité avec les problèmes d'affectations en fait un problème passionnant pour saisir la difficulté intrinsèque de la combinatoire. Nous allons dans un premier temps présenter les problèmes d'affectations ainsi qu'une méthode de résolution appelée « méthode Hongroise ». Dans un second temps, nous allons réutiliser cette méthode afin de résoudre le problème du voyageur de commerce via l'algorithme de little, basé sur les principes de séparation et évaluation.

2 Problèmes d'affectations

2.1 Modélisation

Un problème d'affectation peut être représenté par un matrice de coûts dans laquelle il faut choisir un unique élément par ligne et par colonne, tout en minimisant la somme des éléments choisis. La matrice de coût représente l'ensemble des choix possible pour affecter une ressource (lignes) à une tâche (colonne). Le problème consiste donc à optimiser la répartition des ressources disponibles aux taches à exécuter au moindre coût. Les domaines d'application sont nombreux et rassemblent l'ensemble des problèmes modélisables par cette matrice de coût entre des ressources et des tâches, comme par exemple la répartition des employés (ressources) au travail (tâches). Il s'agit également d'un cas particulier des problèmes de transport dans lesquels la demande associée à chaque destination vaut 1.

2.2 Méthode Hongroise

Afin de résoudre les problèmes d'affectations, nous disposons de l'algorithme de Kühn, également appelé méthode Hongroise.

2.2.1 Lemme pour la méthode hongroise

2.2.1.1 Enoncé

Dans un problème d'affectation, quand on ajoute une constante dans une colonne de la matrice de coût ou dans une ligne de la matrice de coût alors on a les propriétés suivantes :

1. La valeur de la fonction objectif est augmentée de cette constante
2. La solution optimale reste la même

2.2.1.2 Preuve

Soit c une constante réelle et j_0 un indice de colonne. On note $x_{i,j}$ la variable qui vaut 1 la ressource i est affectée à la tâche j, 0 sinon. On note $c_{i,j}$ le coût de réalisation de la tâche j par la ressource i.

La valeur de la fonction objectif est donnée par $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}.x_{i,j}$.

Si on modifie la colonne de la matrice de coût en ajoutant la constante c à tous les éléments de la colonne alors la fonction objectif devient :

- $\sum_{j=1,j\neq j_0}^{n}\sum_{i=1}^{n} c_{i,j}.x_{i,j} + \sum_{i=1}^{n}(c_{i,j_0}+c).x_{i,j_0} = \sum_{j=1,j\neq j_0}^{n}\sum_{i=1}^{n} c_{i,j}.x_{i,j} + \sum_{i=1}^{n} c_{i,j_0}.x_{i,j_0} + c\sum_{i=1}^{n} x_{i,j_0}$
- $\sum_{j=1,j\neq j_0}^{n}\sum_{i=1}^{n} c_{i,j}.x_{i,j} + \sum_{i=1}^{n}(c_{i,j_0}+c).x_{i,j_0} = \sum_{j=1,}^{n}\sum_{i=1}^{n} c_{i,j}.x_{i,j} + c\sum_{i=1}^{n} x_{i,j_0}$

Or dans un programme d'affectation, une tâche est affectée une et une seule fois, on en déduit : $\sum_{i=1}^{n} x_{i,j_0} = 1$. On en conclut que la nouvelle valeur de la fonction objectif est égale à $\sum_{j=1,}^{n}\sum_{i=1}^{n} c_{i,j}.x_{i,j} + c$. En d'autre terme la nouvelle valeur de

la fonction objectif est égale à l'ancienne valeur augmentée de la constante (1ère partie du lemme). Donc cette nouvelle valeur est indépendante du programme d'affectation et ainsi la solution optimale est la même (2ème partie du lemme).

2.2.2 Dynamique de la méthode Hongroise

L'algorithme est composé des 3 étapes détaillées ci-après :

2.2.2.1 Réduction de la matrice initiale

Considérons la matrice de coût suivante :

17	15	9	5	12
16	16	10	5	10
12	15	14	11	5
4	8	14	17	13
13	9	8	12	17

On soustrait à chaque ligne de la matrice le plus petit élément de la ligne, puis on soustrait à chaque colonne le plus petit élément de la colonne.

Ceci nous donne la matrice suivante :

Réduction des lignes :

12	10	4	0	7	(-5)
11	11	5	0	5	(-5)
7	10	9	6	0	(-5)
0	4	10	13	9	(-4)
5	1	0	4	9	(-8)

Puis réduction des colonnes :

12	9	4	0	7
11	10	5	0	5
7	9	9	6	0
0	3	10	13	9
5	0	0	4	9
	(-1)			

2.2.2.2 Couverture des 0

On détermine ensuite le nombre minimal de lignes nécessaires (lignes et colonnes) afin de couvrir tous les 0. Si ce nombre est égal au nombre de lignes (ou colonnes), la matrice est réduite et l'algorithme termine. Sinon on passe à l'étape suivante.

12	9	4	**0**	**7**
11	10	5	**0**	**5**
7	9	9	**6**	**0**
0	3	10	**13**	**9**
5	0	0	**4**	**9**

Il faut 4 lignes pour couvrir tous les 0 donc on continue

2.2.2.3 Mise à jour des lignes et colonnes

Si la matrice n'est pas encore réduite, on recherche la valeur minimale non couverte, on la retranche des autres cases non couvertes et on l'ajoute aux cases situées à l'intersection des lignes de couverture. On recommence ensuite l'étape précédente.

12	9	4	**0**	**7**
11	10	5	**0**	**5**
7	9	9	**6**	**0**
0	3	10	**13**	**9**
5	0	0	**4**	**9**

La valeur minimale non couverte est 3, on effectue la réduction et on recommence l'étape précédente:

12	6	1	**0**	**7**	
11	7	2	**0**	**5**	
7	6	6	**6**	**0**	
0	0	7	**13**	**9**	
8	**0**	**0**	**7**	**12**	**(+3)**
	(-3)	(-3)			

3 Problème du voyageur de commerce

3.1 Modélisation

Le problème du voyageur de commerce (TSP) consiste à trouver un circuit hamiltonien dans un graphe orienté valué représentant des distances entre villes. La version la plus courante, minTSP consiste à trouver le circuit de coût minimum, mais on peut également citer les versions maxTSP ou min-metric TSP, bien que nous ne les étudieront pas dans ce cours.

3.2 Complexité

Le problème minTSP appartient aux problèmes dit complexes de la classe de complexité NP, pour laquelle aucun algorithme efficace n'a été trouvé. Les seuls algorithmes exacts (garantissant la minimalité de la solution) connus jusqu'à présent sont donc tous de complexité exponentielle (non efficaces aux limites). Deux approches s'offrent alors à nous, soit flirter avec les limites afin de garantir la minimalité, soit rechercher des solutions approchées via des algorithmes efficaces.

3.3 Méthodes exactes

La première méthode exacte à considérer est bien entendu la recherche exhaustive consistant à rechercher tous les circuits hamiltoniens et à garder celui de cout minimum. Cette méthode atteint très rapidement des limites non exploitables car de complexité en O(n!). A titre d'exemple pour 20 villes, cela nécessite 20!=2 432 902 008 176 640 000, soit plus de 2 milliard de milliard d'instructions machines. Même avec les meilleurs calculateurs actuels cette limite reste inatteignable. Deux améliorations permettent de réduire drastiquement cette limite bien que restant exponentiels. La

première utilise un algorithme de programmation dynamique que l'on doit à M. Held et M. Karp en 1962, qui résout minTSP en $O(n^2*2^n)$, ce qui ramène notre problème avec 20 villes dans des limites acceptables : 419 430 400 instructions machines. La seconde utilise un algorithme de séparation et évaluation que l'on doit à J.D.C Little, K.G. Murty, D.W. Sweeney et C. Karel en 1963. Bien que sa complexité soit difficile à déterminer, il s'agit à l'heure actuelle de la solution exacte la plus efficace expérimentalement, certains problèmes de plusieurs centaines de villes ayant été résolus en l'utilisant. C'est donc cette méthode que nous allons développer dans la suite de ce cours.

3.4 Méthodes approchées

Une autre solution, que nous ne détaillerons pas dans ce cours consiste à utiliser des méthodes approchées efficaces. Dans ce cas nous n'avons plus de problèmes aux limites, mais aucune garantie d'optimalité ne peut être avancée. L'algorithme basé sur l'arbre couvrant de poids minimal vu l'an passé en cours de graphe est une méthode de ce type et garantie simplement de ne pas dépasser d'un facteur 2 le coût de la solution optimale. D'autres approches basées sur des algorithmes heuristiques (recuit simulé, recherche taboue ou algorithmes génétiques) sont également possibles (cf cours optimisation non déterministe des GSI), mais n'offrent que des approximations qu'il faut vérifier expérimentalement.

4 Algorithme de Little

4.1 Description

L'algorithme de Little est constitué des étapes suivantes :

4.1.1 Calcul des regrets de la matrice et réduction

Cette étape consiste à évaluer, comme dans les problèmes de transport du chapitre précédent, les regrets associés aux lignes et aux colonnes. On commence par calculer les regrets associés aux lignes, ce qui correspond à la somme des valeur minimales de chaque ligne.

Calcul des regrets sur les lignes: $R_1=1+3+1+4+1=10$

X1	X2	X3	X4	X5	
--	11	1	7	9	X1
5	--	3	12	3	X2
7	1	--	9	13	X3
14	9	5	--	4	X4
3	12	7	1	--	X5

On réduit ensuite la matrice via la première étape de la méthode hongroise

X1	X2	X3	X4	X5	
--	10	0	6	8	X1
2	--	0	9	0	X2
6	0	--	8	12	X3
10	5	1	--	0	X4
2	11	6	0	--	X5

On calcul à nouveau les regrets, cette fois sur les colonnes : R2= 2+0+0+0+0=2

On obtient par réduction les matrices de coût réduite et une première borne estimant le coût minimal : $C=R_1+R_2=10+2=12$

X1	X2	X3	X4	X5	
--	10	0	6	8	X1
0	--	0	9	0	X2
4	0	--	8	12	X3
8	5	1	--	0	X4
0	11	6	0	--	X5

4.1.2 Choisir la case nulle dont le coût d'éviction est maximal.

Le coût d'éviction d'une case de la matrice correspond à la somme minimale des cases différentes sur la même ligne et sur la même colonne. Elle représente le surcoût minimal engendré par le non choix d'une case. On choisit donc la case nulle (valeur 0) dont le coût d'éviction est maximal.

X1	X2	X3	X4	X5	
--	10	0 (6)	6	8	X1
0 (0)	--	0 (0)	9	0 (0)	X2
4	0 (9)	--	8	12	X3
8	5	1	--	0 (1)	X4
0 (0)	11	6	0 (6)	--	X5

4.1.3 Suppression de la ligne et de la colonne correspondantes.

X1	X3	X4	X5	
--	0	6	8	X1
0	0	9	0	X2
8	1	--	0	X4
0	6	0	--	X5

4.1.4 Rendre infini le coût de retour (on met un --).

X1	X3	X4	X5	
--	0	6	8	X1
0	--	9	0	X2
8	1	--	0	X4
0	6	0	--	X5

4.1.5 Recommencer avec la matrice partielle obtenue.

X1	X3	X4	X5	
--	0 (7)	6	8	X1
0 (0)	--	9	0 (0)	X2
8	1	--	0 (1)	X4
0 (0)	6	0 (6)	--	X5

X1	X4	X5	
--	9	0 (9)	X2
8	--	0 (8)	X4
0 (8)	0 (9)	--	X5

X1	X4	
8	--	X4
--	0	X5

4.1.6 Arborescence

On présente l'algorithme sous forme d'une arborescence permettant l'évaluation de tous les chemins dont le coût est inférieur au premier coût final trouvé.