GAUTHIER
Thibault
M1 Informatique

Professeur : HAO

Projet Graphe

30 octobre 2013

Problème de « clique »
Langage Java

Enseignant : M. Jin Kao HAO

Année 2013-2014

Travail réalisé par Thibault GAUTHIER

Table des matières

GAUTHIER Thibault
M1 Informatique

Professeur : HAO

Projet Graphe

30 octobre 2013

Introduction

Un graphe est un ensemble de sommets et d'arêtes. Une arête est une paire de sommets qui sont reliés entre eux. Nous travaillons ici sur des graphes non orientés, c'est-à-dire qu'il est possible de passer du sommet s au sommet t et du sommet t au sommet s par la même arête. Dans un graphe non orienté, une clique est un ensemble de sommets dont le sous-graphe engendré par cet ensemble est un graphe complet, c'est-à-dire que tous les sommets sont reliés deux à deux par une arêtes. Une clique peut-être maximale, elle n'est alors comprise dans aucune autre clique, ou maximum, elle est alors la clique du graphe qui possède la plus grande cardinalité (le plus de sommets).

Le but de ce projet est de trouver une solution approchée de la clique maximum d'un graphe. Ce problème est un problème NP-complet. Les algorithmes connus pour résoudre ce problème possède un temps d'exécution qui est exponentiel au nombre de sommet du graphe et au nombre d'arêtes de ce graphe. Une solution pour améliorer ce problème est de mettre en place un algorithme glouton. Un algorithme glouton est un algorithme qui, à chaque étape, effectue un choix optimum local pour espérer trouver une solution optimum globale.

Choix de la modélisation

Il existe deux grandes possibilités pour représenter un graphe : la matrice d'adjacence et la liste d'adjacence. Ces deux possibilités possèdent chacune des avantages et des inconvénients.

La matrice d'adjacence

La matrice d'adjacence permet la mise en place d'une matrice n*n, où n est le nombre de sommets du graphe. Chaque cellule (x,y) contient soit la valeur 0, lorsqu'il n'existe pas d'arête entre les sommets x et y, soit la valeur 1 si l'arc existe. Le principal avantage de cette modélisation est le temps constant de l'opération pour tester l'existence d'une arête. En effet, il suffit de vérifier si la cellule (x,y) de la matrice contient la valeur 1. Mais cette modélisation possède également un inconvénient non négligeable : le gaspillage de mémoire. Chaque cellule de la matrice qui possède la valeur 0 est inutile. Dans le cas d'un graphe très peu dense, cette modélisation devient tout de suite inappropriée.

La liste d'adjacence

La liste d'adjacence permet de représenter, pour chaque sommet, uniquement la liste des sommets adjacents à ce sommet. Cette modélisation, contrairement à la précédente, ne gaspille pas de mémoire. En revanche, le temps nécessaire pour tester l’existence d'une arête augmente en fonction du nombre de sommet et de la densité du graphe. En effet, pour tester l'existence de l'arête (x,y), il faut parcourir la liste d'adjacence de x et vérifier qu'elle contient y. Le temps de parcours est donc fonction de la taille de liste.

Le choix

Étant donné que la recherche de la plus grande clique se fait en testant l'existence de très nombreuses arêtes, la modélisation qui a été retenue est la matrice d'adjacence. Le langage a été retenu pour le développement de ce projet.

Présentation de l'algorithme glouton

Un algorithme glouton a été mis en place pour ce projet. Ce même algorithme propose deux choix optimum local différents. Cet algorithme parcourt tous les sommets du graphe. Pour chaque sommet, il créer une liste des sommets adjacents à ce sommet. Si la taille de la liste trouvée est supérieure à la taille de la clique de référence, il trie les sommets de la liste, selon un critère optimum local que nous détaillerons ensuite, et créer une clique temporaire avec le sommet actuel. Une fois la liste des sommets triée, l'algorithme parcourt cette liste et pour chaque sommet, essaie d'ajouter le sommet à la clique temporaire. Si la taille de clique temporaire est supérieure à la taille de clique de référence, l'algorithme recopie la clique temporaire dans la clique de référence.

L'algorithme

```
entier sizePossible = 0, sizeReal = 0, sizeMax = 0 ;
List<entier> temp , toCheck, maxClique ;
Pour i allant de 0 à n-1 (Où n est l'ordre du graphe)
|       sizePossible = sizeReal = 0 ;
|       vider(temp), vider(toCheck) ;
|       ajouter(temp, i) ;
|       Pour j allant de 0 à n-1
|       |       Si (i,j) est une arête
|       |       |       sizePossible = sizePossible + 1 ;
|       |       |       ajouter(toCheck, j) ;
|       |       FinSi
|       FinPour
|       Si sizePossible est supérieur à sizeMax
|       |       // Trie de la liste toCheck suivant l'optimum choisi.
|       |       Pour tous les entiers m de toCheck
|       |       |       booléen test = vrai ;
|       |       |       Pour tous les entiers n de temp
|       |       |               Si (m,n) n'est pas une arête
|       |       |                       test = false ; break ;
|       |       |               FinSi
|       |       |       FinPour
|       |       |       Si test est vrai
|       |       |       |       ajouter(temp, m) ;
|       |       |       |       sizeReal++ ;
|       |       |       FinSi
|       |       FinPour
```

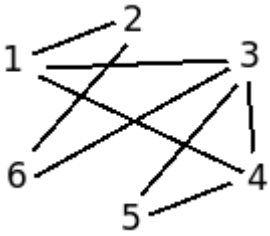
| **FinSi**
| **Si** sizeReal > sizeMax
| | sizeMax = sizeReal
| | vider(maxClique)
| | copier(temp, maxClique)
| **FinSi**
FinPour

Différents optimum local

Différents optimum local sont disponibles pour trier la liste des sommets adjacents à chaque sommet.

Le premier est un optimum aléatoire puisqu'il trie aléatoirement la liste des adjacents. Le second trie les sommets en fonction du nombre de sommets adjacents qu'il possède eux-même. Enfin, le troisième trie les sommets en fonction du nombre de sommets adjacents qu'ils possèdent dans la liste elle même.

Exemple :



Nous cherchons dans ce graphe la plus grande clique contenant le sommet 1.

La liste d'adjacence de 1 est la suivante : 2,3,4.

Pour l'algorithme 1, les sommets seront classés dans l'ordre suivant : 3,4,2.

Pour l'algorithme 2, ils seront classés dans cet ordre : 3,4,1.

Explications :

Pour l'algorithme 1, tous les sommets adjacents sont pris en compte, le sommet 3 a donc 4 adjacents (1,4,5,6), le sommet 4 en a 3 (1,3,5) et le sommet 2 en a 2 (1,6).

Pour l'algorithme 2, nous retirons pour chaque sommet les adjacents qui ne sont pas dans la liste d'adjacence de 1. Le sommet 3 possède donc 2 adjacents (1,4), le sommet 4 en possède 2 également (1,3) tandis que le sommet 2 n'en possède qu'un seul (1).

Tableau des résultats

Nom	M	S	A	D	Tc	R1	T1	R2	T2
brock200_2.clq	12	200	9876	0,496	120	10	80	10	94
brock800_4.clq	26	800	207643	0,650	94	19	127	18	1434
C125.9.clq	34	125	6963	0,898	116	32	61	33	65
C2000.9.clq	80	2000	1799532	0,900	675	63	983	63	28330
C2000.5.clq	16	2000	999836	0,500	367	14	429	15	15088
C4000.5.clq	18	4000	4000268	0,500	1388	15	2289	16	115678
DSJC500_5.clq	13	500	125248	0,502	35	12	30	12	246
DSJC1000_5.clq	15	1000	499652	0,500	90	14	101	13	1931
gen200_p0.9_44.clq	44	200	17910	0,900	25	35	15	36	38
gen400_p0.9_75.clq	75	400	71820	0,900	40	47	32	48	263
hamming8-4.clq	16	256	20864	0,639	20	16	6	16	52
hamming10-4.clq	40	1024	434176	0,829	152	32	107	32	3176
keller4.clq	11	171	9435	0,649	44	11	5	10	23
keller6.clq	59	3361	4619898	0,818	1415	41	1796	44	90233
MANN_a27.clq	126	378	70551	0,990	34	125	22	125	228
MANN_a81.clq	1100	3321	5506380	0,999	1665	1096	12160	1096	138831
p_hat300-3.clq	36	300	33390	0,744	124	32	100	32	299
p_hat1500-3.clq	94	1500	847244	0,754	296	81	428	85	10673

Résultats observés

Légende :

M : Meilleure clique connue.

S : Nombre de sommets du graphe.

A : Nombre d'arêtes du graphe.

D : Densité du graphe.

Tc : Temps de chargement du graphe.

R1 : Meilleure clique trouvée par l'algorithme 1.

T1 : Temps d'exécution de l'algorithme 1.

R2 : Meilleure clique trouvée par l'algorithme 2.

T2 : Temps d'exécution de l'algorithme 2.