Dorian COFFINET
Mickaël FARDILHA

Projet Graphe M1 Informatique

UE3 - Modèles des graphes et de l'intelligence artificielle

Professeur référent : Jin-Kao Hao

Master Informatique promotion 2013-2014
Université d'Angers

I) Présentation de l'application

1) Environnement de développement

Nous avons choisi de développé l'application en Java 7. Nous avons choisi de coder dans ce langage car nous sommes tout deux familiarisé avec Java depuis quelques années. Nous avons codé avec l'IDE Eclipse Kepler 4.3.1. Il nous apparu essentiel de coder avec un IDE. Celui ci nous apporte de nombreuses fonctionnalités pour le développement de telle application.
Un autre point important nous avons utilisé un logiciel de versionning pour l'ensemble du projet ainsi qu'un dépôt en ligne. Nous avons utilisé Git et SVN pour le logiciel de versionning et GitHub pour le répertoire en ligne. Nous avons l'habitude d'utiliser ces outils, il nous a parus naturel de les utiliser dans ce projet.

2) Architectures

a) Architecture du projet

Nous avons fait le choix de découper le code en plusieurs package pour mieux se repérer dans notre projet :

- Package algorithme, contient les deux algorithmes de recherche de clique maximal.
- Package context, contient la représentation du graphe à l'aide des classes Edge, Sommet et Graphe.
- Package doc, contient l'ensemble des documents que nous avons utilisé pour mener à bien le projet. Comme par exemple des fichiers sources de graphe ou des tests
- Package FileRW, contient la classe permettant de lire un fichier
- IG, contient la classe main de l'application et permet de lancer l'application graphique

b) Structure de modélisation du graphe

Deux choix s'offrait à nous, soit modéliser le graphe avec une matrice d'entier ou d'utiliser des listes. Nous avons choisi de travailler avec les listes car Java fourni de différentes structures de listes.
Initialement, nous avions choisi de mettre en place la structure suivante :
- Classe Sommet
- Classe Edge
- Classe Graphe

Étant donné que le but du projet est d'observer les performances, nous avons pris la décision de ne pas créer d'objets superflus tel "Sommet" ou "Edge". Ainsi, nous nous sommes tourné vers une classe unique Graphe, contenant une ArrayList<ArrayList<Integer>>. Cette structure modélisait une liste de sommet contenant, pour chaque sommets, ses arcs.

Au terme de cette seconde implémentation, nous observions des temps corrects, mais peu convainquant. C'est pourquoi nous avons décidé, après avoir optimiser au maximum les créations, instantiations et parcours de variables, de modifier la structure de notre classe Graphe.
Nous avons donc choisi de transformer notre ArrayList<ArrayList<Integer>> en TreeMap<Integer, ArrayList<Integer>)
Cette map permet de stocker sous la forme suivante les informations tirées de la lecture du fichier choisi :

- Clef : Valeur du sommet
- Valeur : Liste des sommets adjacents relatif au sommet en clef

Ainsi, après avoir mis en place cette seconde optimisation de l'algorithme, nous avons pu observer un gain de temps (environs 5 fois plus rapide qu'avec la structure initiale).

Au terme de cette troisième version offrant des temps relativement corrects, nous nous sommes aperçu que nous étions bridé en ce qui concerne les performance, à cause de la structure en objet JAVA. Ainsi, nous avons opté pour la mise en place d'un traitement multi-threadé, en fonction du nombre de coeur de la machine exécutant le programme.
Nous avons donc appliqué la règle suivante : si la machine possède au moins 4 coeurs (logiques et physiques confondus), nous découpons la recherche de la clic maximum en fonction du nombre de coeurs.
Grâce à cela, nous avons pu observer une fois une plus une diminution du temps d'exécution significative : sur une machine possédant 4 coeurs, l'algorithme est au moins trois fois plus efficace.

3) Algorithme utilisé

a) Présentation générale des algorithme

Nos deux algorithme fonctionne suivant les schémas ci dessous. Pour le sommet de départ(en jaune) nous itérons sur chaque sommet du graphe. Ainsi nous augmentons nos chances de trouver la clique maximal. A chaque tour le résultat sera différent.
A chaque itération nous avons 3 paquet de sommet :

1. Paquet comprenant tous les sommets du graphe (paquet vert).
2. Paquet comprenant les sommets formant la clique maximal trouvé jusqu'à présent (paquet bleu).
3. Paquet comprenant l'ensemble des sommets formant une clique avec le paquet bleu (paquet gris).

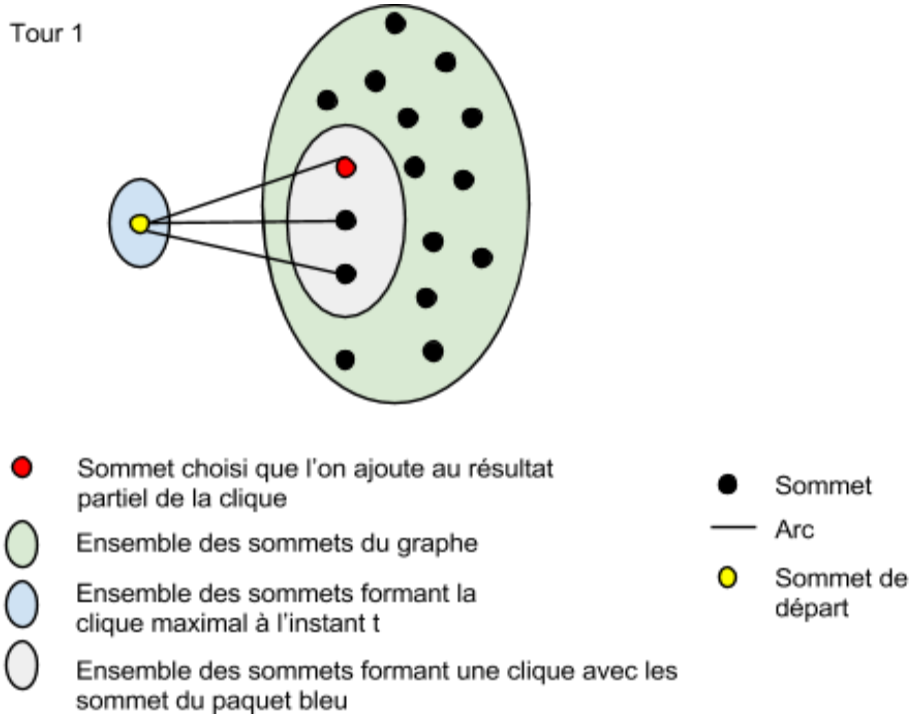


Schéma de la première itération de l'algorithme

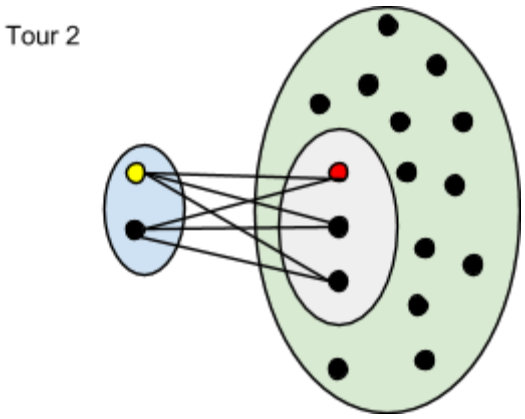


Schéma de la seconde itération de l'algorithme

On répète l'opération jusqu'à trouver la clique maximum depuis notre sommet de départ.

Nos deux algorithmes diffèrent sur le choix du sommet que l'on ajoute à l'ensemble bleu.

b) Algorithme 1

Notre premier algorithme va prendre le premier sommet trouvé qui forme une clique avec les sommets de l'ensemble bleu. Des que l'on trouve on passe à l'itération suivante.

c) Algorithme 2

Notre second algorithme va prendre le sommet trouvé qui forme une clique avec les sommets de l'ensemble bleu et qui a le plus grand nombres de sommets adjacents. Des que l'on trouve on passe à l'itération suivante.

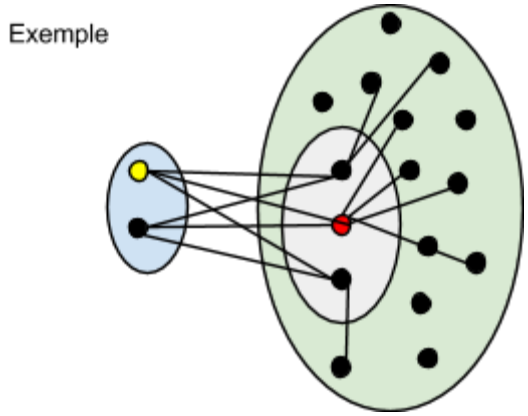


Schéma de choix de sommet avec l'algorithme 2

Ici les sommets qui forment une clique avec l'ensemble bleu ont respectivement, 2, 4 et 1 sommets adjacents. Nous allons donc ajouter le sommet, en rouge, qui a 4 sommets adjacents et passer à l'itération suivante.

II) Tableau de résultats

Avant de ce lancer dans les tests des graphes donnés dans la littérature nous avons créé deux petit graphe l'un avec une clique maximal de 3 avec 5 sommets et 6 arcs et un second graphe avec une clique maximal de 4 avec 5 sommets et 8 arcs.

a) Résultats

Le tableau ci dessous répertorie les résultats de nos deux algorithmes avec les graphes donnés par la littérature. L'ensemble des tests sont fait à partir d'éclipse à l'aide d'un processeur 4 coeurs, de ce fait en changeant de machine les temps peuvent varier.

La 1ère colonne correspond au nom du graphe, la 2ème la clique maximal donnée par la littérature, la 3ème la clique maximal trouvée par notre algorithme 1, la 4ème le temps d'exécution de l'algorithme 1, la 5ème la clique maximal trouvé par, notre algorithme 2, la 6ème le temps d'exécution de l'algorithme 2.

Le temps est exprimé en seconde.
Lorsque l'algorithme prend trop de temps (> à plusieurs heures) on note N.D.

Nom graphe	Clique Maximal Littérature	Algorithme 1	Temps	Algorithme 2	Temps
C125.9	34	29	0,08	31	0,2
C250.9	44	35	1,3	40	5,1
C500.9	57	42	6,7	47	35,3
C1000.9	68	52	78,5	57	346,6
C2000.9	80	56	980,9	N.D.	N.D.
DSJC1000_5	15	12	1,7	14	6,1
DSJC500_5	13	11	0,3	12	1,8
C2000.5	16	14	21	N.D.	N.D.
C4000.5	18	N.D.	N.D.	N.D.	N.D.
MANN_a27	126	39	3,6	125	143,1
MANN_a45	345	66	172,3	N.D.	N.D.
MANN_a81	1100	N.D.	N.D.	N.D.	N.D.
brock200_2	12	10	0,06	10	0,09
brock200_4	17	14	0,08	14	0,1
brock400_2	29	20	0,6	22	2,5
brock400_4	33	21	0,6	21	4,1
brock800_2	24	17	3,1	19	9,7
brock800_4	26	17	2,6	19	10,6
gen200_p0.9_44	44	32	0,6	35	2,1
gen200_p0.9_55	55	35	0,5	39	2,6
gen400_p0.9_55	55	39	7,7	46	27,9
gen400_p0.9_65	65	40	3,5	43	19,5
gen400_p0.9_75	75	42	4,6	47	18,1
hamming10-4	40	32	59,3	32	99,8
hamming8-4	16	16	0,2	16	0,5
keller4	11	9	0,02	11	0,01
keller5	27	22	4,8	21	15,9
keller6	59	N.D.	N.D.	N.D.	N.D.
p_hat300-1	8	7	0,02	8	0,05
p_hat300-2	25	19	0,5	24	1
p_hat300-3	36	25	0,7	32	3,4
p_hat700-1	11	9	0,6	9	0,4
p_hat700-2	44	30	7,7	41	39,5
p_hat700-3	62	43	11,3	57	115,7
p_hat1500-1	12	9	1,7	10	6,2
p_hat1500-2	65	39	93,3	59	846,6
p_hat1500-3	94	61	357,1	N.D.	N.D.

Tableau des résultats des algorithmes 1 et 2

b) Interprétation des résultats

Premièrement l'algorithme 1 plus léger permet de l'exécuter sur des graphes plus important tout en restant dans un ordre de temps raisonnable (inférieur à plusieurs heures)

Secondement regardons les résultats par type de graphe.
Pour les algorithmes de type C:

- Les résultats de l'algorithme 1 sont inférieurs à ceux de l'algorithme 2
- Plus le nombre d'arc augmente, plus les résultats trouvés, des deux algorithme, sont éloignés du résultat optimal.

Pour les algorithmes de type DSJC :

- Les résultats de l'algorithme 1 sont inférieurs à ceux de l'algorithme 2
- Les algorithme trouve des résultats proches du résultat optimal

Pour les algorithmes de type MANN:

- L'algorithme 2 trouve un résultat très proche du résultat optimal
- L'algorithme 1 fonctionne assez mal sur ce type de graphe

Pour les algorithmes de type brock:

- Les résultats entre l'algorithme 1 et 2 sont très proches, bien que l'algorithme 2 trouve quelques fois un résultat plus grand
- Plus le nombre d'arc augmente, plus les résultats trouvés, des deux algorithme, sont éloignés du résultat optimal.

Pour les algorithmes de type gen:

- Les résultats de l'algorithme 1 sont inférieurs à ceux de l'algorithme 2
- Les deux algorithme trouve un résultat très loin du résultat optimum

Pour les algorithmes de type hamming:

- Les résultats sont identiques entre l'algorithme 1 et 2

Pour les algorithmes de type keller:

- Les résultats de l'algorithme 1 sont inférieurs à ceux de l'algorithme 2 pour keller4
- Par contre pour keller5, l'algorithme 1 trouve un résultat plus proche de l'optimum que l'algorithme 2
- Avec le graphe keller4 l'algorithme 2 parvient à trouver le résultat optimum

Pour les algorithmes de type p_hat

- Les résultats de l'algorithme 1 sont inférieurs à ceux de l'algorithme 2
- Avec les petits graphes (p_hat300-1, p_hat300-2…) le résultat de l'algorithme 2 est très proche voir égale à la solution optimale

c) Synthèse

Globalement l'algorithme 1 trouve des résultats moins bon que l'algorithme 2. Mais l'algorithme 1 s'exécute beaucoup plus vite et permet de trouver un résultat sur des graphes avec de nombreux sommets dans un temps raisonnable.
Le tableau suivant montre l'efficacité des deux algorithmes suivant les types de graphe.
Nous allons noter les algorithmes bon quand les résultats sont proches du résultat optimal, moyen pour des résultats moyen et mauvais pour des résultats loin du résultat optimal sans tenir compte du temps d'exécution.

Type Graphe	Algorithme 1	Algorithme 2
C	Moyen	Moyen
DSJC	Moyen	Bon
MANN	Mauvais	Bon
brock	Moyen	Moyen
gen	Mauvais	Mauvais
hamming	Bon	Bon
keller	Bon	Bon
p_hat	Bon	Bon

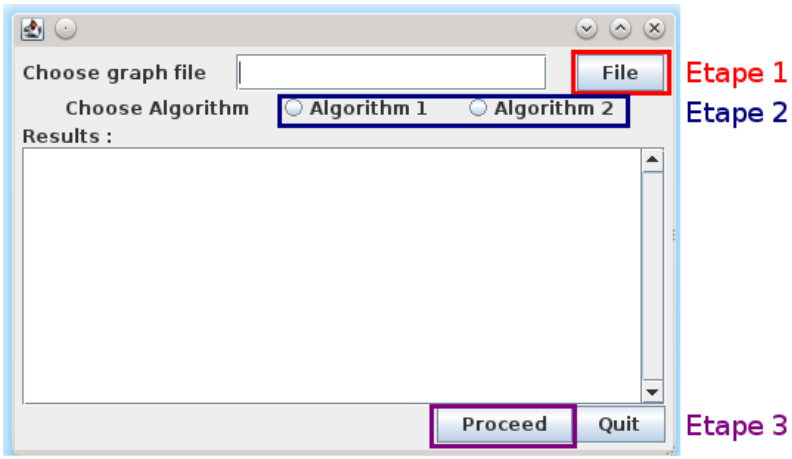
Tableau de synthèse des résultats

III) L'interface Graphique

L'interface graphique permet d'utiliser nos algorithmes plus facilement. Une fois lancée :

1. Choisir le fichier via le file chooser
2. Choisir l'algorithme 1 ou 2
3. Cliquer sur le bouton "Proceed"

Des lors le programme se lance et affiche la clique maximum trouvée dans la zone de texte "résultat".



L'exécution de l'algorithme est bloquant pour l'application. Tant que l'algorithme n'est pas fini le bouton "Proceed" reste bloqué. Une fois l'algorithme fini le résultat apparaît et on peut faire une nouvelle exécution d'algorithme.

III) Amélioration et critique de l'application

a) Critiques

Nous avons choisi d'utiliser des structures de listes dans un premier temps. Nous nous sommes aperçu que la résolution de nos algorithmes augmente plus il y a d'arc dans le graphe. Nous avons donc opté pour utiliser les threads pour diminuer le temps d'exécution. Nous avons grandement diminué le temps d'exécution. Par exemple pour le graphe du fichier C1000.9.clq , nous sommes passé de plusieurs heures à environ 1h30 pour avoir le résultat avec l'algorithme 2.
Pour essayer de gagner encore du temps nous avons essayé de passer à une structure avec des tableaux. Néanmoins, modifier deux algorithmes basés sur les ArrayList et les TreeMap se trouve être bien compliqué. En effet, nous profitons des avantages de ces objets JAVA : une ArrayList est un tableau dynamique, ce qui se révèle être un véritable atout pour un tel développement. Passer nos algorithmes sous forme de tableaux est périlleux dès lors qu'il faut prendre garde à gérer correctement la tailles de ces derniers.

b) Passage à une matrice

Après réflexion nous pensons que le passage par matrice aurait été une bonne solution pour diminuer notre temps d'exécution. Car un ne manipule plus de structure de liste mais uniquement un tableau à double entrée.
Nous aurions pu créer d'autres algorithmes gloutons. Nous avons opté pour deux choix de sommets mais il y en a beaucoup d'autre. Il aurai été possible de choisir le sommet au hasard parmi l'ensemble de sommets qui forment une clique.