Université de Bordeaux
Département Licence

université de BORDEAUX

Implémentation d'algorithmes de reconnaissance de graphes d'intervalles

Stage effectué au LaBRI
sous la tutelle de M BAUDON
Du 15 Mai au 16 Juin 2014

CONSTANS Olivier

3ème Année de licence Informatique
2013-2014

Table des matières

Remerciements

Tout d'abord, je tiens à remercier tout particulièrement et à témoigner ma reconnaissance à M Baudon, mon tuteur de stage pour l'aide qu'il m'a apporté pendant mon stage. Sa disponibilité, sa pédagogie, ses remarques et ses conseils m'ont été utiles dans la réalisation de mon projet de stage.

Chapitre 1

Introduction

1.1 Cadre du stage

Dans le cadre de la formation en Licence Informatique, les étudiants sont amenés à effectuer un stage d'une durée minimum d'un mois, ceci afin de permettre aux étudiants de se familiariser avec le monde du travail et avoir un expérience professionnelle qui les conforteraient dans leur choix d'orientation en master.

Mon stage s'est déroulé au Laboratoire Bordelais de Recherche en Informatique (LaBRI) au sein de l'équipe "Combinatoire et Algorithmique". Il a été effectué sous la tutelle de M Baudon.

Le LaBRI est une unité de recherche associée au CNRS (UMR 5800), à l'Université de Bordeaux et à l'IPB. Depuis 2002, il est partenaire de l'INRIA. Ses effectifs se sont accrus de façon importante ces dernières années. En mars 2014, il réunit près de 320 personnes, dont 112 enseignants chercheurs (Université de Bordeaux, IPB), 37 chercheurs (CNRS, INRIA), 23 personnels administratifs et techniques (Université de Bordeaux, IPB, CNRS, INRIA) et plus de 140 doctorants,post-doctorants et ingénieurs contractuels. Les missions du LaBRI s'articulent autour de trois axes principaux : recherche (théorique, appliquée), valorisation - transfert de technologie et formation.

Le laboratoire s'articule autour de six équipes thématiques alliant recherche fondamentale, recherche appliquée et transfert technologique :

- Combinatoire et Algorithmique
- Méthodes Formelles
- Modèles et Algorithmes pour la Bioinformatique et la Visualisation d'informations
- Programmation, Réseaux et Systèmes

– Supports et Algorithmes pour les Applications Numériques hautes performances
– Image et Son

Chaque équipe est répartie en thèmes. M Baudon travaille dans le thème "Graphes et applications" de l'équipe "Combinatoire et Algorithmique". Voici les deux autres thèmes de cette équipe :

– Algorithmique distribuée
– Combinatoire énumérative et algébrique

1.2 Sujet du stage

Le sujet de mon stage est de rajouter des algorithmes pour la reconnaissance des graphes d'intervalles dans une bibliothèque déjà existante en Java de M. Baudon.

Durant ce stage, il m'a donc fallu comprendre certaine notions et algorithmes de la théorie des graphes nécessaires pour la réalisation de mon stage.

Parallèlement, j'ai dû me familiariser avec la bibliothèque de M. Baudon pour pouvoir par la suite implémenter les algorithmes.

Dans la deuxième partie, je vous présente mon travail réalisé durant mon stage. Et dans la troisième partie, je fait un bilan et une conclusion de mon stage et de son déroulement.

Chapitre 2

Travail réalisé

2.1 Notions de la théorie des graphes

Pour pouvoir comprendre mon travail réalisé durant mon stage, il faut connaître certaines notions de la théorie des graphes. Voici ci-dessous les notions de la théorie des graphes nécessaires pour la compréhension de mon travail réalisé. Les définitions et théorèmes m'ont été donné par M. Baudon, pour plus d'information on pourra consulter par exemple les livres de M. Distel [7] ou de M. Golumbic [6].

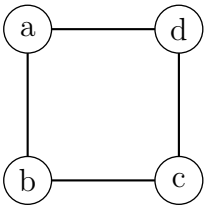
Définition 1 *Un* graphe G *est un couple* (V, E) *formé de deux ensembles finis* $V = \{v_1, \ldots, v_n\}$ *et* $E = \{e_1, \ldots, e_m\}$*, et où pour tout* i*,* e_i *est composée de deux éléments de* V*, pas forcément distincts.* V *est appelé l'ensemble des* sommets *et* E *l'ensemble des* arêtes*.*

Deux arêtes sont dites *parallèles* si elles ont les mêmes extrémités. Un ensemble d'arêtes parallèles est appelé *arête multiple* . Une arête reliant un sommet à lui-même est appelée une *boucle.*

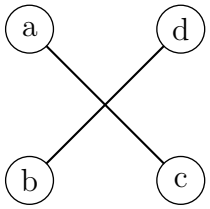
Définition 2 *Un graphe est dit simple s'il est sans boucle, ni arête multiple.*

Pour la suite, nous n'allons seulement nous intéresser aux graphes simples.

Définition 3 *Le graphe complémentaire d'un graphe simple* G *est un graphe simple* H *ayant les mêmes sommets et tel que deux sommets distincts de* H *soient adjacents si et seulement si ils ne sont pas adjacents dans* G*.*

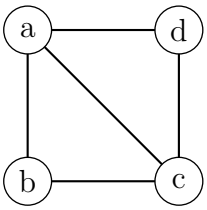


(a) Graphe G

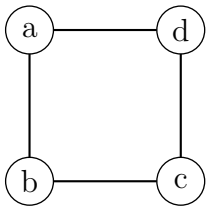


(b) Complémentaire du graphe G

Définition 4 *Un graphe est triangulé (ou chordal en anglais) si tout cycle de longueur supérieur ou égale à 4 possède une corde, c'est à dire une arête reliant deux sommets non consécutifs dans le cycle.*

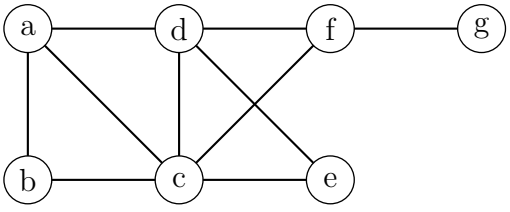


(c) Graphe triangulé

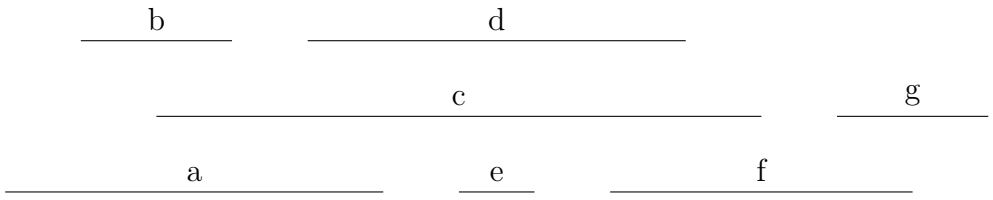


(d) Graphe non-triangulé

Définition 5 *Un graphe d'intervalles est le graphe d'intersection d'intervalles sur une droite. Donc, étant donné un ensemble $I=\{I_1,\ldots,I_n\}$ d'intervalles sur une droite, on lui associe le graphe $G=(V,E)$ où $V=I$ et deux sommets I_i et I_j sont reliés par une arête si et seulement si $I_i \cap I_j \neq \emptyset$.*



(e) Graphe d'intervalle



(f) Intervalles associé au graphes

Définition 6 *Un graphe G(V,E) est de comparabilité si il admet une orientation F transitive c'est à dire une orientation des arêtes telle que :*
$uv \in F$ *et* $vw \in F \Rightarrow uv \in F$

Définition 7 *Une clique d'un graphe non orienté est un sous-ensemble des sommets de ce graphe qui induit un sous-graphe complet, c'est-à-dire que deux sommets quelconques de la clique sont toujours adjacents.*

2.2 Méthode utilisant les graphes de comparabilité

Dans un premiers temps, M. Baudon m'a demandé de réaliser l'algorithme de reconnaissance des graphes d'intervalles en utilisant le propriété suivante :

Théorème 1 *G est un graphe d'intervalle si et seulement si G est triangulé et son complémentaire est un graphe de comparabilité.[1]*

Ainsi, je devais implémenter trois algorithmes :
- un algorithme de reconnaissance des graphes triangulés.
- un algorithme donnant le complémentaire d'un graphe.
- un algorithme de reconnaissance des graphes de comparabilité.

2.2.1 Reconnaissance d'un graphe triangulé

Les graphes triangulés sont caractérisés par l'existence d'un schéma d'élimination simplicial. On dit qu'un sommet est simplicial si son voisinage induit un sous-graphe complet (une clique). Les graphes triangulés peuvent donc être construit (ou réduit) en ajoutant (ou supprimant) un à un des sommets simpliciaux.

Nous avons utilisé l'algorithme LexBFS qui construit un ordre d'élimination. Cet ordre est simplicial si et seulement si le graphe est triangulé. Puis nous vérifions que l'ordre d'élimination donné par LexBFS est simplicial ou non.

Données:
Un graphe G = (V,E)
Résultat:
Un ordre λ. λ est un ordre d'élimination simplicial λ si et seulement si G est triangulé
début
pour chaque *sommet* $x \in V$ **faire**
étiquette(x) = $\emptyset$
pour $i = n$ *jusqu'à* 1 **faire**
Choisir un sommet non numéroté x $\in V$ d'étiquette maximum
λ(i) $\leftarrow$ x
pour chaque *voisin non numéroté de y de* $x \in V$ **faire**
Ajouter i à étiquette(y)
fin

Algorithme 1: LexBFS [2]

Données:
Un graphe G = (V,E)
Résultat:
Retourne si G est un graphe triangulé.
début
pour chaque $x \in V$ *dans l'ordre LexBFS* **faire**
si *les voisins de x sont une clique* **alors**
Supprimer le sommet x de G.
sinon
retourner faux
retourner vrai
fin

Algorithme 2: Vérification que l'ordre LexBFS est simplicial

Exemple 1

Soit le graphe G :

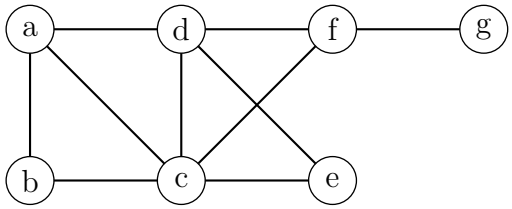


FIGURE 2.1 – Graphe G

On applique le LexBFS sur G :

	a	b	c	d	e	f	g
init	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
a	$\emptyset$	7	7	7	$\emptyset$	$\emptyset$	$\emptyset$
b	$\emptyset$	7	76	7	$\emptyset$	$\emptyset$	$\emptyset$
c	$\emptyset$	7	76	75	5	5	$\emptyset$
d	$\emptyset$	7	76	75	54	54	$\emptyset$
e	$\emptyset$	7	76	75	54	54	$\emptyset$
f	$\emptyset$	7	76	75	54	54	2
g	$\emptyset$	7	76	75	54	54	2

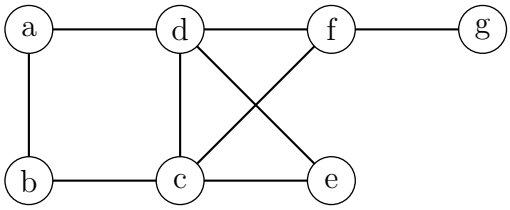
Ainsi l'ordre LexBFS est g,f,e,d,c,b,a. Maintenant on vérifie si l'ordre est simplicial.

x	voisinage de x
g	$\{f\}$ est une clique
f	$\{d, c\}$ est une clique
e	$\{d, c\}$ est une clique
d	$\{a, c\}$ est une clique
c	$\{a, b\}$ est une clique
b	$\{a\}$ est une clique
a	$\emptyset$ est une clique

Donc G est un graphe triangulé.

Exemple 2

Soit le graphe G' :



On applique le LexBFS sur G' :

	a	b	c	d	e	f	g
init	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
a	$\emptyset$	7	$\emptyset$	7	$\emptyset$	$\emptyset$	$\emptyset$
b	$\emptyset$	7	6	7	$\emptyset$	$\emptyset$	$\emptyset$
d	$\emptyset$	7	65	7	5	5	$\emptyset$
c	$\emptyset$	7	65	7	54	54	$\emptyset$
e	$\emptyset$	7	65	7	54	54	$\emptyset$
f	$\emptyset$	7	65	7	54	54	2
g	$\emptyset$	7	65	7	54	54	2

Ainsi l'ordre LexBFS est g,f,e,c,d,b,a. Maintenant on vérifie si l'ordre est simplicial.

x	voisinage de x
g	$\{f\}$ est une clique
f	$\{d, c\}$ est une clique
e	$\{d, c\}$ est une clique
c	$\{b, d\}$ n'est pas une clique

Donc G' n'est pas un graphe triangulé.

2.2.2 Création du complémentaire

Pour construire le complémentaire d'un graphe G, il suffit de prendre tout les couples de sommets du graphe G et vérifier leur adjacence. Si les 2 sommets ne sont pas adjacents dans G, il faut rajouter une arête entre les deux sommets dans G'.

2.2.3 Reconnaissance d'un graphe de comparabilité

Après quelques recherches sur le sujet, nous avons trouvé un article sur l'orientation transitive de Mc. Cornell et M. Spinrad [3]. Mais j'ai eu du mal à comprendre leur algorithme. Nous avons finalement préféré changer de méthode pour reconnaître un graphe d'intervalles. Nous avons suivi la méthode présentée par Christophe Paul et Laurent Viennot [2].

2.3 Méthode utilisant les arbres de cliques

M. Baudon et moi avons trouvé un article de Christophe Paul et Laurent Viennot [2]. Cet article traite des utilisations de l'algorithme LexBFS. Un des algorithmes proposé permet de reconnaître un graphe d'intervalles par une autre caractérisation que l'utilisation des graphes de comparabilité :

Théorème 2 *Un graphe d'intervalles est un graphe triangulé dont l'arbre de cliques est une chaîne. [2]*

Autrement dit, il existe un ordonnancement des cliques maximales tel que les cliques contenant un sommet donné soient consécutives. On appelle un tel arrangement, une chaîne de cliques. Un graphe d'intervalles peut posséder plusieurs chaînes de cliques. Trouver une chaîne de cliques permet de reconnaître un graphe d'intervalle et d'identifier les intervalles.

Ainsi, deux algorithmes devaient être implémentés :

- un algorithme de reconnaissance de graphe triangulé (le même que dans la méthode utilisant les graphes de comparabilité) ;
- un algorithme vérifiant l'existence d'une chaîne de cliques.

Pour vérifier l'existence d'une chaine de cliques, il faut préalablement calculer un arbre de cliques ce qui être obtenue en modifiant LexBFS.

2.3.1 Construction de l'arbre de cliques

Dans l'article de de Christophe Paul et Laurent Viennot, ils proposent un algorithme permettant de construire l'arbre de cliques en utilisant LexBFS (algorithme déjà étudié durant le stage pour prouver qu'un graphe est triangulé).

Données:
Un graphe G = (V,E).
Résultat:
Si G est triangulé : un ordre d'élimination simplicial et un arbre de clique T = (I,F).
début

pour chaque *sommet* $x \in V$ **faire**
étiquette(x) = $\emptyset$

étiquette_précédente = $\emptyset$
j = 0
pour *i = n jusqu'à 1* **faire**
Choisir un sommet non numéroté x $\in V$ d'étiquette maximum
si *étiquette_précédente* $\notin$ *étiquette(x)* **alors**
j = j+1
Créer la clique maximale C_j = étiquette(x) $\cup \{x\}$
C(dernier(x)) est le père de C_j dans T
L'arc de l'arbre C_jC(dernier(x)) est étiqueté par le séparateur minimal :
$S_j = C_j \cap$ C(dernier(x)) = étiquette(x)
sinon
$C_j = C_j \cup \{x\}$

pour chaque *voisin non numéroté de y de* $x \in V$ **faire**
Ajouter i à étiquette(y)
dernier(y) = x

étiquette_précédente = étiquette(x)
λ(i) $\leftarrow$ x
C(x) = j

fin

Algorithme 3: Lex-BFS et arbre de cliques [2]

Grâce à cet algorithme nous obtenons un arbre de cliques et l'ordre de découverte des cliques par LexBFS, nécessaire pour la recherche d'une chaîne de cliques et la reconnaissance d'un graphe d'intervalle.

Exemple 1

Appliquons l'algorithme au graphe G de la figure 2.1 :

	a	b	c	d	e	f	g	n° C	C_1	C_2	C_3	C_4	C_5
init	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$						
a	$\emptyset$	7	7	7	$\emptyset$	$\emptyset$	$\emptyset$	1	$\{a\}$				
b	$\emptyset$	7	76	7	$\emptyset$	$\emptyset$	$\emptyset$	1	$\{a, b\}$				
c	$\emptyset$	7	76	75	5	5	$\emptyset$	1	$\{a, b, c\}$				
d	$\emptyset$	7	76	75	54	54	$\emptyset$	2	$\{a, b, c\}$	$\{a, c, d\}$			
e	$\emptyset$	7	76	75	54	54	$\emptyset$	3	$\{a, b, c\}$	$\{a, c, d\}$	$\{c, d, e\}$		
f	$\emptyset$	7	76	75	54	54	2	4	$\{a, b, c\}$	$\{a, c, d\}$	$\{c, d, e\}$	$\{c, d, f\}$	
g	$\emptyset$	7	76	75	54	54	2	5	$\{a, b, c\}$	$\{a, c, d\}$	$\{c, d, e\}$	$\{c, d, f\}$	$\{f, h\}$

On obtiens l'arbre suivant :

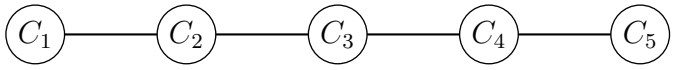


FIGURE 2.2 – Arbre de cliques du graphe G

Exemple 2

Appliquons l'algorithme au graphe H :

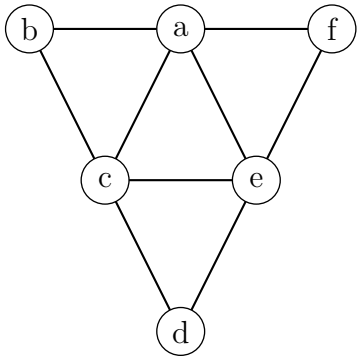


FIGURE 2.3 – Graphe H

	a	b	c	d	e	f	n° C	C_1	C_2	C_3	C_4
init	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$					
a	$\emptyset$	6	6	$\emptyset$	6	6	1	$\{a\}$			
b	$\emptyset$	6	65	$\emptyset$	6	6	1	$\{a, b\}$			
c	$\emptyset$	6	65	4	64	6	1	$\{a, b, c\}$			
e	$\emptyset$	6	65	43	64	63	2	$\{a, b, c\}$	$\{a, c, e\}$		
f	$\emptyset$	6	65	43	64	63	3	$\{a, b, c\}$	$\{a, c, e\}$	$\{c, e, f\}$	
d	$\emptyset$	6	65	43	64	63	4	$\{a, b, c\}$	$\{a, c, e\}$	$\{c, e, f\}$	$\{c, e, d\}$

On obtiens l'arbre suivant :

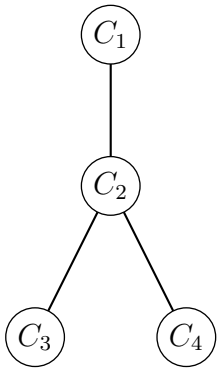


FIGURE 2.4 – Arbre de cliques du graphe H

2.3.2 Reconnaissance d'un graphe d'intervalle

L'algorithme de reconnaissance partitionne l'ensemble des cliques en s'appuyant sur un arbre de cliques. La partition de départ est constituée de la dernière clique visitée par LexBFS et des autres cliques. Dans une chaîne de cliques, les cliques contenant un sommet donné sont consécutives : chaque raffinement du partitionnement sépare la clique courante des cliques ne contenant pas un sommet pivot par les cliques qui le contiennent.

Données:
Un graphe G = (V,E).
Résultat:
Si G est un graphe d'intervalle : une chaîne de cliques L.
début

Calculer les cliques maximales et un arbre de clique T = (I,F).
si *G n'est pas triangulé* **alors**
retourner "G n'est pas un graphe d'intervalle"

Soit I l'ensemble des cliques maximales $I = \{C_1,...,C_k\}$
Soit L la liste ordonnée (I)
pivots = $\emptyset$ est un pile vide
tant que *il existe une classe I_c qui n'est pas un singleton dans $L = (I_1,....,I_l)$* **faire**
si *pivots* = $\emptyset$ **alors**
Soit C_l la dernière clique de I_c découverte par Lex-BFS
Remplacer I_c par $I_c \setminus \{C_l\}$, $\{C_l\}$dans L
$C = \{C_l\}$
sinon
Choisir un sommet x non utilisé de pivots (supprimer ceux qui ont été utilisé)
Soit C l'ensemble de toutes les cliques contenant x

si *toutes les cliques de C apparaissent dans des classes consécutives* **alors**
Soit I_a la première classe contenant une telle clique
Soit I_b la dernière classe contenant une telle clique
sinon
retourner "G n'est pas un graphe d'intervalle"

si *Une classe strictement entre I_a et I_b contient une clique n'appartenant pas à C* **alors**
retourner "G n'est pas un graphe d'intervalle"

Remplacer I_a par $I_a \setminus C$, $I_a \cap C$ et I_b par $I_b \cap C$; $I_b \setminus C$
pour chaque *arc C_iC_j de l'arbre de cliques connectant une clique $C_i \in C$ à une clique $C_j \notin C$* **faire**
pivots = pivots $C_i \cap C_j$
Supprimer $C_iC_j$de l'arbre de cliques

fin

Algorithme 4: Partitionnement des cliques maximales [2]

Pour rendre plus lisible les cliques, nous noterons la clique $\{a, b, c\}$ connectant les sommets a, b, c sous la forme abc .

Exemple 1

Appliquons l'algorithme au graphe G de la figure 2.1 :
G est bien triangulé.
L'arbre de cliques T est celui de la figure 2.2 et l'ordre LexBFS est g,f,e,d,c,b,a.

pivot	L	I_a	I_b
init	$\{\{dfc, dec, dac, abc, fg\}\}$		
$\emptyset$	$\{\{dfc, dec, dac, abc\}, \{fg\}\}$	$\{fg\}$	$\{fg\}$
f	$\{\{dec, dac, abc\}, \{dfc\}, \{fg\}\}$	$\{dfc, dec, dac, abc\}$	$\{fg\}$
c	$\{\{dec, dac, abc\}, \{dfc\}, \{fg\}\}$	$\{dec, dac, abc\}$	$\{dfc\}$
d	$\{\{abc\}, \{dec, dac\}, \{dfc\}, \{fg\}\}$	$\{dec, dac, abc\}$	$\{dfc\}$
a	$\{\{abc\}, \{dec, dac\}, \{dfc\}, \{fg\}\}$	$\{abc\}$	$\{dac, dec\}$

pivot	C	empiler
$\emptyset$	$\{fg\}$	f
f	$\{dfc, fg\}$	c, d
c	$\{dfc, dec, dac, abc\}$	c, d
d	$\{dfc, dec, dac\}$	a, c
a	$\{dac, abc\}$	a, c

L ne possède que des singletons, donc G est un graphe d'intervalles.

Exemple 2

Appliquons l'algorithme au graphe H de la figure 2.3 :
G est bien triangulé.
L'arbre de cliques T est celui de la figure 2.4 et l'ordre LexBFS est d,f,e,c,b,a.

pivot	L	I_a	I_b	C	empiler
init	$\{\{fa, dec, eac, abc\}\}$				
$\emptyset$	$\{\{efa, dec, eac\}, \{abc\}\}$	$\{abc\}$	$\{abc\}$	$\{abc\}$	a, c
c	$\{\{efa\}, \{dec, eac\}, \{abc\}\}$	$\{efa, dec, eac\}$	$\{abc\}$	$\{dec, eac, abc\}$	e, a
a	$\{\{efa\}, \{dec, eac\}, \{abc\}\}$	$\{efa\}$	$\{abc\}$	$\{eac, efa, abc\}$	

Mais il existe une classe $\{dec, eac\}$ strictement entre I_a et I_b contenant une clique $\{dec\}$ n'appartenant pas à C. Le graphe H n'est donc pas un graphe d'intervalles.

Chapitre 3

Conclusion

3.1 Travail réalisé

Les algorithmes présentés plus haut ont été implémentés dans la bibliothèque Java de M Baudon, dans un fichier nommé "Stage.java". Ainsi si mon travail ne satisfait pas totalement M Baudon, il est simple de le supprimer ou de le modifier avant de l'intégrer réellement dans la bibliothèque, en rajoutant le contenu du fichier "Stage.java" dans le fichier "Graphes.java".

Le fichier "Stage.java" contient les fonctions suivantes :

- Graph<V, E> complement(Graph<V, E> g) : retourne le complémentaire du graphe g.

- boolean isClique(Graph<V, E> g, Set<V> e) : qui vérifie si un ensemble de sommets e est bien une clique dans le graphe g.

- List<V> lexBFS(Graph<V, E> g) : retourne un ordre d'élimination.

- boolean isTriangulated(Graph<V, E> g) : vérifie si g est un graphe triangulé.

- LexBFSResults<V> lexBFSAndCliqueTree(Graph<V, E> g) : retourne un ordre d'élimination et un arbre de clique. L'ordre d'élimination est simplicial si g est triangulé.

- boolean isPerfectEliminationOrdering(Graph<V, E> g, List<V> l) : vérifie si la liste de sommet l est un ordre d'élimination simplicial dans le graphe g.

– Graph<Set<V>, Graph.Edge<Set<V>>> cliqueTree(Graph<V, E> g) : retourne l'arbre de clique du graphe G.

– boolean isIntervalGraph(Graph<V, E> g) : vérifie si le graphe G est un graphe d'intervalles.

De plus des fichiers de test et des tests unitaires ont été ajoutés pour vérifier le bon fonctionnement des fonctions réalisées.

3.2 Répartition du travail

La durée du stage était de 4 semaines. La première semaine ainsi que la moitié de la seconde ont consisté à prendre en main la bibliothèque Java de M. Baudon et à la tentative de réalisation de l'algorithme de reconnaissance d'un graphe d'intervalles en se servant des graphes de comparabilité. La semaine et demie suivante a servi à la réalisation de l'algorithme de reconnaissance grâce aux arbres de clique. Durant cette période, la majeure partie de temps a été utilisée dans la compréhension des algorithmes, alors que peu de temps, on suffit pour leur implémentation. La dernière semaine du stage, ainsi que les deux semaines suivantes, ont servi à l'élaboration du rapport du stage et à la prise en main des outils Latex nécessaire pour sa réalisation.

3.3 Bilan

J'ai rencontré quelques difficultés durant le stage. La compréhension de nouveaux algorithmes s'est montré plus difficile que ce que j'imaginais. Cela m'a permis de mettre en évidence un de mes défauts, la précipitation.

Lors de ces 4 semaines de stage au sein du LaBRI, j'ai pu mettre en pratique et consolider certaines de mes connaissances théoriques acquises durant mes années de formation de licence, notamment mes connaissances en Java et sur la théorie des graphes. J'ai appris de nouveaux algorithmes et de nouvelles notions comme les graphes d'intervalles ou les graphes de comparabilité. De plus j'ai eu l'occasion de m'initier aux tests unitaires avec JUnit. Ces nouvelles connaissance seront un atout pour ma future formation et lors de ma vie professionnel.

Le travail demandé a été réalisé, à la fois en terme de programmation et de documentation (présent rapport).

Bibliographie

[1] P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16 :539–548, 1964. doi:10.4153/CJM-1964-055-5.

[2] C. Paul and L. Viennot. Quelques algorithmes linéaires de reconnaissance autour de Lex-BFS. http://hal.inria.fr/docs/00/47/16/10/PS/ami96.ps.

[3] R McConnell and J.P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of undirected graphs. In *Proc. 7th Annual ACM-SIAM Symp. Discrete Algorithm*, pages 19–35. SIAM, Philadelphia, 1997.

[4] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoritical Computer Science*, 234 :59–84, 2000.

[5] P. Galinier, M. Habib, and C. Paul. Chordal graphs and their clique graph. In M. Nagl, editor, *Graph-Theoric Concepts in Computer Science*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371, Aachen, Germany, 1995. WG '95, Springer, Berlin.

[6] M.C. Golumbic. Algorithmic graph theory and perfect graphs. In *Annals of Discrete Mathematics*. Elevier, 2004.

[7] R. Destiel. Graph theory. In *Graduate Texts in Mathematics*. Springer, 2005.