N° d'ordre : 15 / 2005 – M/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE

Université des Sciences et de la Technologie Houari Boumediene

Faculté de Mathématiques

Mémoire

Présenté pour l'obtention du diplôme de Magister
EN MATHEMATIQUES

Spécialité : Recherche Opérationnelle (Mathématiques de Gestion)
Par : BOUTICHE Mohamed Amine

SUJET :

GRAPHES ET PROBLEMES DE ROUTAGE

Soutenu publiquement le : 22 – 10 – 2005, devant le jury composé de :

Abdelhafid	BERRACHEDI	Professeur, USTHB	Président
Hacène	AIT HADDADENE	MC, USTHB	Directeur de thèse
Méziane	AIDER	Professeur, USTHB	Examinateur
Isma	BOUCHEMAKH	MC, USTHB	Examinateur
Mourad	BOUDHAR	MC, USTHB	Examinateur

Remerciements

Je tiens à remercier Mr. AIT HADDADENE (mon promoteur) qui a su me guider et m'orienter dans la réalisation de cette thèse.

Je remercie Mr. BERRACHEDI pour l'honneur qu'il me fait d'être le président du jury.

Je remercie aussi les membres du jury ; Mr. AIDER, Mr. BOUDHAR et Mlle. BOUCHEMAKH pour m'avoir honoré en acceptant de juger mon travail.

Enfin, je remercie mes parents et mes amis qui m'ont soutenu afin d'élaborer ce modeste travail.

Dédicaces

A ma mère et mon père,

A mes frères et sœur,

En particulier MALIA.

A tous ceux qui me sont chers,

J’offre ce modeste travail

Résumé

La construction de schémas de routage compact, dans les classes de graphes qui admettent une décomposition arborescente, dont les sacs sont de faible diamètre, en tolérant une erreur additive, a fait apparaître un nouvel invariant des graphes ; La longueur arborescente.
Les différents travaux présentés dans cette thèse concernent le calcul de cet invariant. Notre premier objectif est de voir quelles classes de graphes sont de longueur arborescente bornée par 2 sachant que les graphes triangulés sont de longueur arborescente égale à 1. Une revue des travaux effectués pour le calcul d'un autre invariant « la largeur arborescente », nous a permis de donner quelques éléments de réponse. Notre deuxième objectif est de voir quelles sont les classes de graphes qui admettent une décomposition arborescente de longueur bornée par une constante. Nous avons fait le lien entre la longueur arborescente et les triangulations minimales des graphes, ce qui nous a permis de calculer la longueur arborescente pour certaines classes de graphes.

Résumé

La construction de schémas de routage compact, dans les classes de graphes qui admettent une décomposition arborescente, dont les sacs sont de faible diamètre, en tolérant une erreur additive, a fait apparaître un nouvel invariant des graphes ; La longueur arborescente.
Dans notre thèse, nous donnons une condition suffisante, pour qu'un graphe soit de longueur arborescente au plus deux.
Ensuite, nous montrons que les graphes k-cordaux sont de longueur arborescente exactement $\lceil k/3 \rceil$.

Mots clefs

Routage compact, schémas de routage, décomposition arborescente, longueur arborescente, cordalité.

Abstract

The construction of additive stretched routing schemes for graph classes, which admit tree decomposition with small diameter bags, contribute to the apparition of a new invariant of graphs ; the tree length.
In our thesis, we give a suffisant condition for graphs with tree length at most two.
Then, we show that the tree length of any k- chordal graph is exactly $\lceil k/3 \rceil$.

Key words

Compact routing, routing schemes, tree decomposition, tree length, chordality.

Table des matières

Introduction

L'une des principales missions de la recherche opérationnelle, est d'aider à la modélisation des problèmes pratiques rencontrés dans divers domaines. Et la théorie des graphes est un outil très puissant qui permet d'analyser les problèmes, en fournissant un niveau d'abstraction qui facilite le processus de modélisation.

Dans un réseau d'interconnexion (des stations de travail ou dans une machine parallèle), les composantes ne sont pas toutes reliées entres elles. Il est donc nécessaire de router l'information entre ces composantes : chacune d'entres elles doit être capable de décider localement, avec la connaissance qu'elle a du réseau, vers ou faire transiter le message. Ainsi, lorsque l'on conçoit de telles architectures, une des préoccupations est de choisir une topologie qui rende la circulation de l'information la plus efficace possible.

Le routage consiste à faire communiquer, de la façon la plus efficace possible ces composantes. Les liens de communications sont bidirectionnels. C'est pourquoi, en général notre réseau sera modélisé par un graphe non orienté. Les sommets du graphe représentent les machines et les arêtes représentent les liens de communications.

Ceci fait apparaître la notion de famille de graphes et de schémas de routage spécifique pour chaque famille.

En général, le problème de routage dans les graphes est un problème NP-complet, mais devient polynomial pour les classes de graphes dont la largeur arborescente est bornée.

Des résultats optimaux ont été obtenus dans [38] pour le routage compact selon les plus courts chemins dans les arbres.

D'autres auteurs [33], [34], [35] ont construits des schémas de routage compact pour les graphes triangulés et pour les graphes qui admettent une décomposition arborescente dont les sacs (ou nœuds) sont de diamètre borné.

Lors de ces travaux, il est apparu que pour faire du routage en tolérant une erreur additive (c'est-à-dire en suivant les plus courts chemins plus une certaine erreur), il est plus judicieux de considérer non pas la cardinalité des sacs (la largeur arborescente), mais plutôt leur diamètre. Le diamètre d'un sac étant la plus grande distance entre deux de ces sommets, ce qui a conduit à faire apparaître un nouvel invariant des graphes, la longueur arborescente qui vient compléter celui de la largeur arborescente déjà abondamment étudiée.

Dans cette thèse, nous nous sommes fixés comme objectif d'étudier les problèmes de calcul de la longueur arborescente des graphes.
On savait que les graphes triangulés étaient de longueur arborescente égale à 1, nous avons montré que les graphes faiblement triangulés sont de longueur arborescente bornée par 2. Nous avons aussi fait le lien entre les triangulations minimales des graphe et les graphes qui admettent une décomposition arborescente de diamètre borné par δ. Ce qui nous a permis de calculer la longueur arborescente des graphes *k*-cordaux qui est une classe de graphes très large.

Notre travail est organisé de la façon suivante :

Chapitre 1 : Généralités
L'objet de ce chapitre est de présenter les différentes notions, définitions et résultats en théorie des graphes et connus des lecteurs avertis.

Chapitre 2 : Triangulations minimales et décompositions arborescentes
Dans ce chapitre, nous donnerons la définition d'un graphe triangulé, d'une triangulation d'un graphe, et nous terminerons par des notions moins classiques qui sont essentielles pour nos travaux ; la décomposition arborescente et sa largeur.

Chapitre 3 : Le problème du routage
Dans ce chapitre, nous donnons les principaux résultats sur le routage dans les classes de graphes ; les arbres, les graphes triangulés, et les graphes qui admettent une décomposition arborescente de faible diamètre.

Chapitre 4 : La longueur arborescente

Dans ce chapitre, nous définissons précisément la longueur arborescente. Puis nous montrerons pourquoi ce concept est utile pour les problèmes de routage. Ensuite, nous donnerons les classes de graphes pour lesquels la longueur arborescente a été calculée.

Chapitre 5 : Calcul de la longueur arborescente

Dans ce chapitre, nous présenterons tous les résultats obtenus lors de nos travaux, notamment ; le calcul de la longueur arborescente pour les graphes faiblement triangulés , les graphes *k*- cordaux, et un algorithme qui calcul la longueur arborescente pour les graphes quelconques.

CHAPITRE 1

Généralités

L'objet de ce chapitre est de présenter les différentes notions, définitions et résultats en théorie des graphes et connus des lecteurs avertis.

1.1 Terminologie

La terminologie employée dans cette thèse est inspirée du livre de Claude Berge [5] .

1.1.1 Graphes

Un graphe G sera noté G = (V,E), où V est l'ensemble des sommets et E est l'ensemble des éléments appartenant au multi – ensemble $V \times V=\{uv / u \in V$ et $v \in V\}$. Les éléments de E sont dits arêtes et seront notées uv (ou $\langle u, v \rangle$).

On notera n le nombre de sommets de G et m son nombre d'arêtes.

Une arête uv est une boucle si $u = v$. Tout au long de la thèse nous ne considérerons que des graphes simples, c'est-à-dire sans boucle ni arête multiple, voir un exemple avec la figure 1.1.

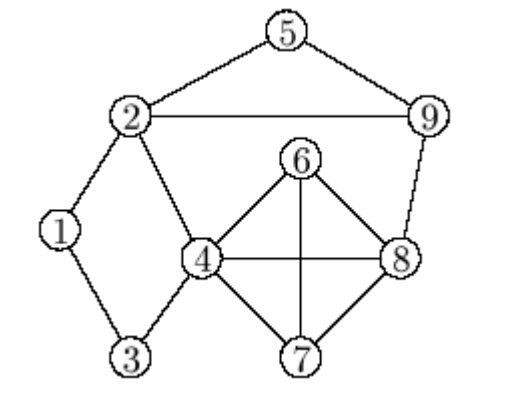


Figure 1.1- Un graphe simple avec 9 sommets et 14 arêtes

Deux sommets u et v sont adjacents s'il existe une arête entre u et v. On dit aussi que v est dans le voisinage du sommet u. On notera $N(u)$ l'ensemble des voisins de u. Le degré d'un sommet u, noté $deg(u)$, désigne le nombre de ses voisins. Sur la figure 1.1 , le sommet 6 est degré 3 car $N(6) = \{4, 7, 8\}$.

Une chaîne dans un graphe G = (V, E), permettant d'aller d'un sommet u_1 à un sommet u_k, est une suite $(u_1, u_2 , ….., u_k)$ de sommets distincts de G telle que pour tout $i \in \{1,…, k-1\}$, (u_i, u_{i+1}) est une arête La longueur d'une chaîne est le nombre d'arêtes de la chaîne, ici k-1.

Un cycle est une chaîne dont le sommet de départ et le sommet d'arrivée sont identiques.
Par convention, on notera par deux fois le sommet extrémité puisqu'un cycle n'a évidemment pas d'extrémités. Une corde dans un cycle est une arête reliant deus sommets non adjacents de ce cycle. Dans le graphe de la figure 1.1, les sommets (1, 2, 5, 9, 8, 7, 4, 3) forment un cycle, les arêtes $\langle 2, 9\rangle$, $\langle 2, 4\rangle$ et $\langle 4, 8\rangle$ en sont des cordes.

Un graphe est connexe, si pour toute paire de sommets *(u , v)*, il existe une chaîne reliant *u* et *v*. Dans un graphe, un séparateur est un sous ensemble de sommets dont la suppression génère un graphe qui n'est plus connexe. Le graphe est alors composé de plusieurs composantes connexes. Un sommet d'articulation est un sommet qui est un séparateur. Un graphe à n sommets (avec $n \geq k + 2$) est k – connexe, s'il faut supprimer au moins k sommets pour qu'il ne soit plus connexe, autrement dit : tout séparateur de G possède au moins k sommets. On utilise aussi le terme biconnexe pour désigner les graphes 2 – connexes.
Le graphe de la figure 1.1, est biconnexe, puisque si l'on supprime n'importe quel sommet, le graphe reste connexe. En revanche ce graphe n'est pas 3 – connexe puisque si l'on supprime les sommets 9 et 4, on obtient deux composantes connexes formées des sommets 1, 2, 3,5 et 6, 7,8.

La distance dans un graphe G entre deux sommets *u* et *v*, notée $dist_G(u,v)$ ou plus simplement *dist(u,v)*, est la longueur d'une plus courte chaîne entre ces deux sommets. Le diamètre de G noté *diam(G)*, est la plus grande distance entre toute paire de sommets. Par exemple le graphe de la figure 1.1 est de diamètre 3.

Le graphe G = (V , E) est dit complet si pour toute paire de sommets *(u , v)* de G, *u* est adjacent à *v*. Un graphe est donc complet si et seulement si son diamètre est 1. (voir figure 1.2).

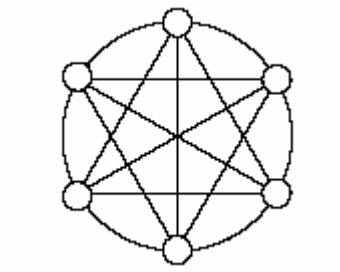


Figure 1.2 – Le graphe complet à 6 sommets

1.1.2 Sous graphes

Un graphe $G' = (V' , E')$ est un sous graphe d'un graphe $G = (V , E)$ si $V' \subseteq V$ et $E' \subseteq E$.
Un sous graphe G' d'un graphe G est un sous graphe induit de G si, pour toute paire de sommets (x, y) de V', $x\,y \in E \Leftrightarrow x\,y \in E'$. Autrement dit, si deux sommets adjacents dans G sont présents dans G', alors ils sont aussi adjacents dans G', G' est appelé le sous graphe de G induit par V', il sera noté $G[V']$.

Un sous graphe G' d'un graphe G est un sous graphe couvrant de G (ou bien sous graphe partiel de G) , si G' contient tous les sommets de G : $V' = V$.

Par exemple, la figure 1.3 montre un sous graphe du graphe de la figure 1.1 qui est induit par les sommets 1,2,3,5,6,7 et 9. Il n'est pas couvrant car il manque les sommets 4 et 8. Il n'est pas non plus connexe car par exemple il ne possède pas de chaîne entre 6 et 3.

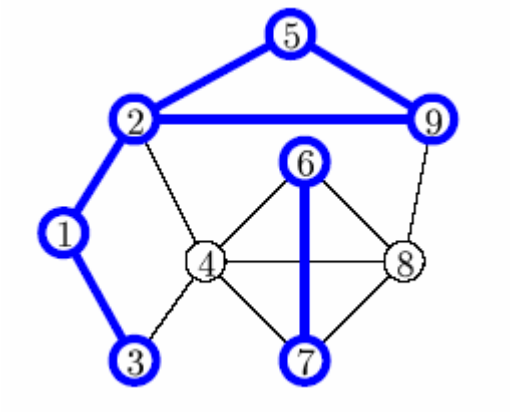


Figure 1.3 – Un sous graphe induit par les sommets 1,2,3,5,6,7 et 9

Pour tout sous graphe G' de G, on définit le diamètre de G' dans G par $diam_G(G') = max_{u,v \in V'}\ dist_G(u,v)$. Bien que la distance entre deux sommets d'un graphe non connexe ne soit pas définie, le diamètre d'un sous graphe non connexe est défini, c'est le maximum parmi les diamètres des composantes connexes du sous graphe. Par exemple, le diamètre du sous graphe représenté sur la figure 1.3 est trois.

Par extension, pour tout sous ensemble $V^{/}$ de sommets de V, on définit le diamètre de $V^{/}$ dans G par : $diam_G(V^{/}) = Max_{C_i \in G[V^{/}]}(diam_G(C_i))$

(avec C_i composante connexe de $G[V^{/}]$ i=1…p et p étant le nombre des composantes connexes de $V^{/}$).

Dans un graphe G, une clique est un sous graphe qui est complet. Une clique maximale est une clique de G qui est maximale par inclusion ; si on ajoute n'importe quel autre sommet de G, on ne peut pas obtenir une clique. On note par ω(G) le nombre de sommet de la plus grande clique de G, qui est aussi appelée la clique maximum de G. Dans le graphe de la figure 1.1 le sous graphe induit par les sommets 2,5 et 9 est une clique maximale qui n'est pas maximum, par contre le sous graphe induit par les sommets 4, 6, 7, 8 est la clique maximum du graphe de la figure 1.1.

1.1.3 Arbres

Un arbre est un graphe connexe sans cycles. Les sommets d'un arbre sont appelés les nœuds de l'arbre. Un arbre est également définit comme un graphe connexe dans lequel il existe une unique chaîne entre toute paire de sommets.

Un arbre enraciné (appelé également arborescence) est un arbre où un nœud est distingué. Ce nœud est appelé racine de l'arbre. Les arbres enracinés seront, lorsque c'est possible, représentés « tête en haut » ; la racine est le sommet situé le plus haut (voir un exemple avec la figure 1.4)

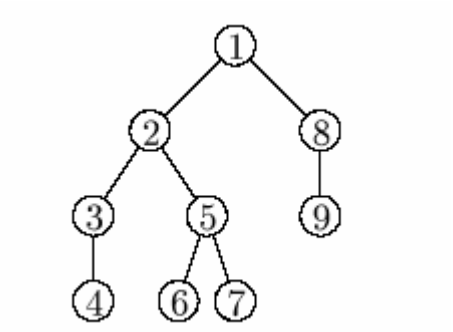


Figure 1.4 – Un arbre enraciné sur un nœud 1

On dit qu'un nœud u est un *ancêtre* d'un nœud v (ou v, un *descendant* de u) s'il se trouve sur la chaîne entre v et la racine de l'arbre. On dit que u est le *parent* de v (ou v, un *enfant*

de u) s'il est l'ancêtre de v qui lui est adjacent. Si deux nœuds ont le même parent, alors ils sont *frères*. Un nœud, autre que la racine , qui possède un ou plusieurs enfants est appelé *nœud interne*. A l'inverse, un nœud qui ne possède pas d'enfant est une *feuille*.
La profondeur d'un nœud u , notée $depth(u)$, est la longueur de la chaîne qui le sépare de la racine. La profondeur d'un arbre est la plus grande des profondeurs de ses feuilles. On appelle *plus petit ancêtre commun* (ou *plus proche ancêtre commun*) entre u et v dans un arbre T, noté $nca_T(u, v)$, le nœud de profondeur maximum qui est à la fois un ancêtre de u , et un ancêtre de v .
Dans l'arbre de la figure 1.4, le nœud 2 a deux enfants : les nœuds 3 et 5, ils sont donc frères. Les nœuds 4,6,7,9 sont des feuilles et l'arbre est de profondeur 3 car les chaînes entre les feuilles 4,6, ou 7 et la racine 1, sont de longueur 3. Le plus petit ancêtre commun entre les nœuds 4 et 7 est le nœud 2.

Un parcours en profondeur d'un arbre (en anglais depth first search : DFS), consiste à traiter la racine de l'arbre puis à parcourir récursivement les sous arbres issus de ses enfants. Un parcours en largeur d'un arbre (en anglais breadth first search : BFS) consiste à parcourir l'arbre par niveau de profondeur : tous les nœuds de profondeur i seront traités avant n'importe quel nœud de profondeur $i + 1$.

Dans l'arbre de la figure 1.4, un parcours en profondeur peut traiter successivement les nœuds 1,2,3,4,5,6,7,8,9 alors qu'un parcours en largeur pourra les traiter dans l'ordre 1,2,8,3,5,9,4,6,7.

1.1.4 Arbres couvrants

Un sous – graphe G' d'un graphe G est un arbre couvrant G si G' couvre G et si G' est un arbre. Un arbre G' couvrant G est appelé un arbre de plus courtes chaînes s'il existe un sommet u tel que pour tout sommet v,

$$dist_G(u,v) = dist_G'(u,v).$$

La figure 1.5 ci – dessous, montre un arbre de plus courtes chaînes du graphe de la figure 1.1.

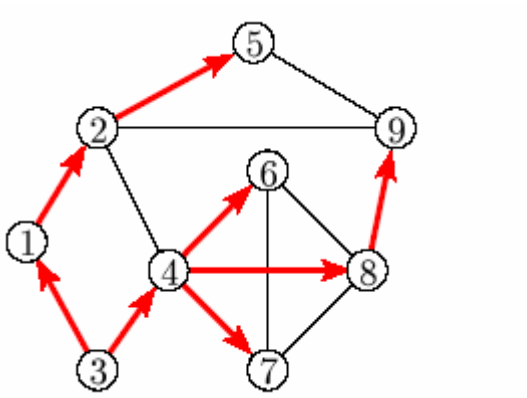


Figure 1.5 – Un arbre couvrant de plus courtes chaînes enraciné sur le nœud 3.

Dans ce cas, l'arbre ne peut pas être représenté « tête en haut », les liens de l'arbre sont alors représentés par des flèches allant d'un nœud vers un de ses enfants. La racine est donc le seul sommet du graphe ne possédant que des flèches sortantes, dans ce cas présent, il s'agit du nœud 3.

Faire un parcours en largeur d'un graphe G, consiste à choisir un sommet (une source), et à parcourir le graphe par niveau d'éloignement de la source : les sommets à distance *i* de la source seront traités avant n'importe quel sommet à distance *i*+1 de la source. Un tel parcours produit un arbre de plus courtes chaînes, enraciné sur le sommet source. Ainsi, un parcours en largeur du graphe de la figure 1.1 peut traiter les sommets dans l'ordre : 3,1,4,2,6,7,8,9,5.

1.2 Complexité Algorithmique

L'expérience montre que certains problèmes sont plus faciles que d'autres à résoudre sur un ordinateur. Une théorie de la complexité a été développée et permet mathématiquement de classer les problèmes faciles et difficiles en deux classes ; la classe P et la classe NP – complet. Nous donnerons dans ce qui suit les principes essentiels de la théorie de la complexité.

Les classes P, NP et NP – complet

Pour pouvoir parler de la notion de classes de problèmes, il est nécessaire de distinguer les problèmes de décision des problèmes d'optimisation. Un problème de décision est un problème pour lequel la réponse est oui ou non. On note qu'il est possible d'associer à chaque problème d'optimisation, un problème de décision en introduisant un seuil k correspondant à

la fonction objectif f . Le problème de décision devient « existe – t – il une solution réalisable (S) telle que $f(S) \leq$ (ou $\geq$) k?

Il est alors possible de définir *la classe P (polynomiale)* qui regroupe les problèmes de décisions résolus par des algorithmes polynomiaux. Un algorithme polynomial est défini comme un algorithme dont le temps d'exécution est en O(p(x)) où p est un polynôme et x est la longueur d'entrée (c'est-à-dire le nombre des données) d'une instance du problème.

La classe NP (Non Deterministic Polynomial) regroupe les problèmes qui peuvent être résolus en temps polynomial par des algorithmes non déterministes (un algorithme est dit non déterministe s'il comporte des instructions de choix). Pour ces algorithmes, si à chaque instruction , le bon choix est effectué, le temps de calcul est polynomial. Si au contraire tous les choix sont énumérés, l'algorithme devient déterministe et son temps de calcul devient exponentiel. De façon informelle, un problème de décision appartient à NP si on peut vérifier en un temps polynomial si une solution « potentielle » donnée satisfait la question posée.

Les algorithmes polynomiaux sont évidement des cas particuliers des algorithmes non déterministes. Aussi tout problème de décision qui peut être résolu par un algorithme polynomial, et qui donc appartient à la classe P, appartient également à la classe NP.

D'où $P \subseteq NP$.

Parmi les problèmes de la classe NP, une large classe des problèmes, les problèmes *NP – complets* sont équivalents entre eux quant à l'existence d'un algorithme polynomial pour les résoudre. C'est-à-dire, s'il existe un algorithme polynomial pour résoudre un seul de ces problèmes, alors il existe un pour chaque problème de la classe NP.

Afin de décrire cette classe d'équivalence, définissons tout d'abord la transformation (réduction) polynomiale entre deux problèmes. Soient D_1 et D_2 deux problèmes de décision. La transformation polynomiale de D_1 vers D_2 (notée $D_1 \alpha D_2$) peut être vue comme une fonction f de l'ensemble des instances de D_1 vers l'ensemble des instances de D_2 qui satisfait aux deux conditions suivantes :

1. f est calculable par un algorithme polynomial.

2. Pour toute instance I de D_1 , I a pour réponse oui (pour D_1) si et seulement si f (I) a pour réponse oui (pour D_2).

D'une manière équivalente, D_1 se transforme (réduit) polynomialement à D_2 , s'il existe un algorithme de résolution de D_1, qui fait appel à un algorithme de résolution de D_2, et qui est polynomial lorsque la résolution de D_2 est comptabilisée comme une opération élémentaire.
On dit alors d'un problème de décision qu'il est NP – complet si tout problème de la classe NP se transforme polynomialement en lui.
Ainsi, il est nécessaire de connaître des problèmes connus pour être NP – complet.
Le premier problème qui a été prouvé comme étant NP – complet, par S.A.Cook (1970) , est le problème de satisfiabilité (SAT) qui peut être défini comme suit : Etant donné une expression booléenne sous forme normale conjonctive (de la forme de conjonctions de disjonctions) , existe –t – il une affectation en « vrai » et « faux » de ses variables de manière à ce que l'expression booléenne prenne la valeur « vrai » ?
Dans le cas où la réponse est oui, l'expression sera dite satisfiable.

CHAPITRE 2

Triangulations minimales et décompositions arborescentes

Dans ce chapitre, nous donnerons les définitions d'un graphe triangulé, d'une triangulation d'un graphe, et nous nous intéresserons à des notions moins classiques et qui sont essentielles pour nos travaux ; les décompositions arborescentes et leurs largeurs.

2.1 Graphes k – cordaux et graphes triangulés

Définition 2.1 *Un graphe est k – cordal s'il ne possède pas de cycle induit de longueur plus grande que k. Ceci signifie que tout cycle de longueur supérieure à k possède une corde. La cordalité d'un graphe G est le plus grand entier k tel que G soit k – chordal.*
Un graphe est triangulé si et seulement si, il est 3 – cordal.

Les graphes triangulés (en anglais *chordal graphs*) ont une structure fortement arborescente. Ils sont souvent considérés comme des arbres généralisés.

Définition 2.2 *Un arbre de cliques est un arbre T dont chaque nœud est une clique maximale de G et satisfaisant la propriété d'intersection des cliques ; « soit A et B deux cliques maximales, l'ensemble A∩B est contenu dans toutes les cliques sur la chaîne reliant A et B dans T ».*

Le théorème suivant est du à Gavril en 1974 :

Théorème 2.1 [41] *Un graphe est triangulé si et seulement si, il admet un arbre de cliques*.

Dans la figure 2.1 ci – dessous, nous présentons un graphe triangulé (tous ses cycles induits sont de longueur 3), son ensemble de cliques maximales et un arbre de clique correspondant.

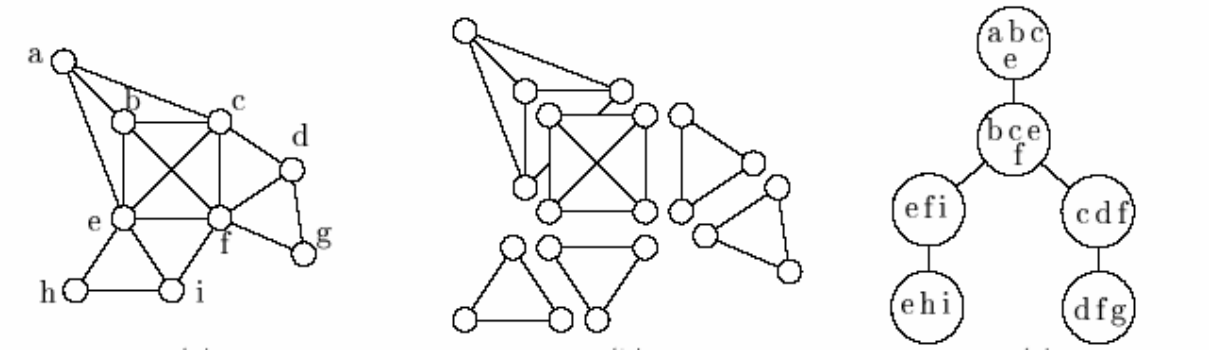


Figure 2.1 – Un graphe triangulé, son ensemble de cliques maximales et un arbre de cliques.

Pour un graphe donné, on peut généralement associer plusieurs arbres de cliques non isomorphes. Cette notion d'arbre de cliques des graphes triangulés sera très utile par la suite pour faire du routage dans les graphes triangulés.
D'autant plus comme nous le montre le théorème suivant, ils sont calculables en temps linéaire :

Théorème 2.2 [32] *Pour tout graphe triangulé G, il est possible de calculer un arbre de cliques de diamètre minimum en temps linéaire.*

Une autre caractérisation des graphes triangulés est donnée par les ordres d'élimination parfaits.

Définition 2.3
Un sommet u est dit simplicial si son voisinage est une clique. Un ordre $x_1, x_2, \ldots, x_n$ *sur les sommets de G tel que* $\forall\ i \in \{1,2,\ldots,n\}$, x_i *est simplicial dans* $G[\{x_i, x_{i+1}, \ldots, x_n\}]$ *est appelé un ordre d'élimination parfait.*

Théorème 2.3 [42] *Un graphe est triangulé si et seulement si, il admet un ordre d'élimination parfait.*

La figure 2.2 ci – dessous présente un ordre d'élimination parfait sur le graphe triangulé de la figure 1.6 : x_1 étant le sommet 1, x_2 étant le sommet 2, . . . x_9 étant le sommet 9.

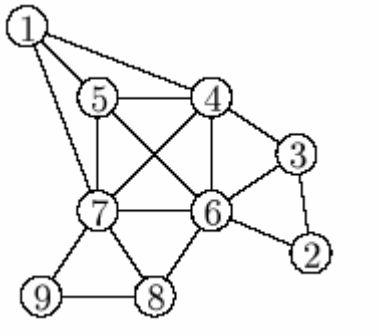


Figure 2.2 – Un ordre d'élimination parfait ($x_i = i$) sur un graphe triangulé.

Cette propriété est à la base des algorithmes tels que LexBFS[55] et MCS[56], qui déterminent, en temps linéaire, si un graphe est triangulé ou non.

2.2 Triangulations et triangulations minimales

Définition 2.3 *Soit $G = (V , E)$ un graphe quelconque. Le graphe $G' = (V' , E')$ est une triangulation de G si G' est un graphe triangulé, si $V = V'$ et si $E \subset E'$.*

Soit $G' = (V , E')$ une triangulation de G, G' est une *triangulation minimale* de G si et seulement si pour tout F tel que $F \subset E'$, le graphe $H = (V , F)$ n'est pas un triangulation de G.

Ainsi dans la figure 2.3 , les trois graphes de droites sont des triangulations minimales du cycle à 6 sommets. En effet ce sont des graphes triangulés, et si on enlève une des arêtes ajoutées, ils ne le sont plus.

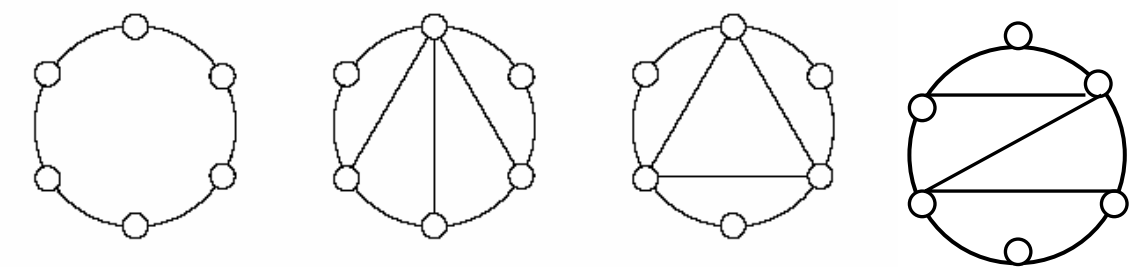


Figure 2.3 – Le cycle à 6 sommets et trois triangulations minimales possibles.

Citons un résultat du à Rose, Tarjan et Lueker en 1976 :

Théorème 2.4 [55] *pour tout graphe G, il est possible de construire en temps O(nm) une triangulation minimale de G.*

2.3 Décomposition arborescente

Nous venons de définir la notion d'arbre de cliques des graphes triangulés, ainsi que celles des triangulations et des triangulations minimales.

Nous allons maintenant pouvoir étendre cette notion d'arbres de cliques aux graphes quelconques en définissant une notion qui sera présente tout au long de cette thèse : la décomposition arborescente.

De plus, avec la notion de séparateurs minimaux, nous présenterons également quelques résultats et problèmes bien connus sur un paramètre essentiel d'une décomposition arborescente : sa largeur.

2.3.1 Définitions

La notion de décomposition arborescente a été introduite par Robertson et Seymour en 1986 lors de leurs travaux sur les mineurs de graphes [54].

Définition 2.5 une décomposition arborescente d'un graphe G = (V , E) est un arbre *T* dont les nœuds, appelés sacs (en anglais bags) sont des sous ensembles de sommets de G. Cet arbre respecte les trois règles suivantes :

1. *pour tout sommet u de G, il existe au moins un sac contenant u.*
2. *pour toute arête (u,v) de E, il existe au moins un sac contenant u et v.*
3. *pour tout sommet u de G, l'ensemble des sacs contenant u induit un sous arbre de T.*

Remarque 2.1

- Notez que la terminologie que nous utiliserons pour parler de la notion de sommets est que si l'on se trouve dans un graphe quelconque nous les désignerons par sommets, si l'on se trouve dans un arbre par nœuds et enfin, si l'on se trouve dans une décomposition arborescente par sacs.

- Une décomposition arborescente d'un graphe G, peut être vue comme un arbre de cliques d'une triangulation particulière de G. De même un arbre de cliques est une décomposition arborescente particulière d'un graphe triangulé. Dans la suite nous noterons donc de la même manière un arbre de clique et une décomposition arborescente : par une lettre italique *T* par exemple.

- Dans la suite, on notera par T_u l'ensemble des sacs de *T* contenant le sommet *u*. Par analogie pour tout sous ensemble *X* de sommets de G, on note T_X l'ensemble des sacs contenant au moins un sommet de *X* : $T_X = \bigcup_{u \in X} T_u$.

- la troisième règle de la définition 2.5 peut s'écrire de manière équivalente par :
Pour tout sacs B_1, B_2, B_3 de *T*, si B_2 est sur la chaîne entre les sacs B_1 et B_3, alors
$B_1 \cap B_3 \subseteq B_2$.

La figure 2.4 présente un graphe G et une décomposition arborescente *T* de G. Il est en effet facile de voir que *T* vérifie les trois règles de la définition 2.5. Cette figure présente aussi la triangulation de G pour laquelle *T* est un arbre de cliques.

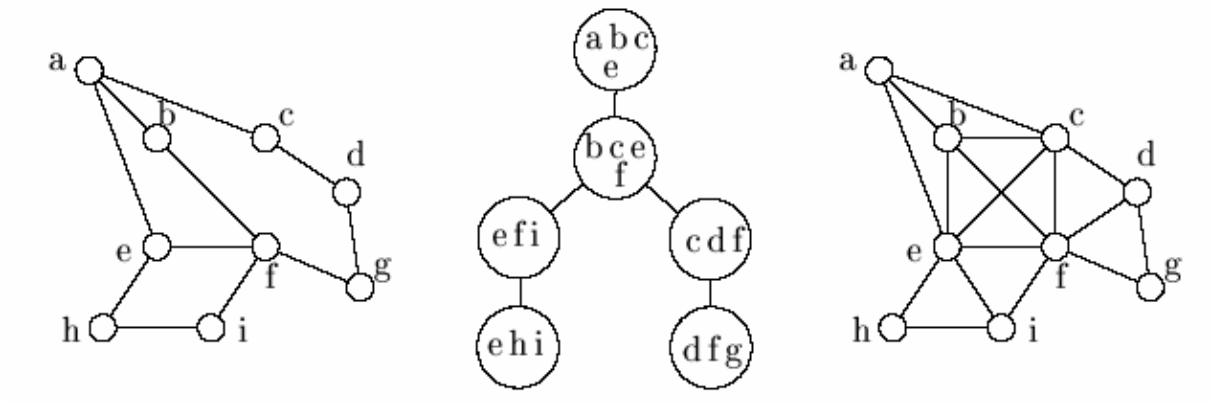


Figure 2.4 – Un graphe G, une décomposition arborescente *T* de G et la triangulation correspondante.

Notons que pour tout graphe, il est très facile de trouver une décomposition arborescente : on prend un arbre avec un seul sac qui contient tous les sommets du graphe. Les trois règles de la définition 2.5 sont alors trivialement respectés.
La proposition suivante présente les propriétés de séparations induites par les décompositions arborescentes :

Proposition 2.1 [24] Soient *T* une décomposition arborescente d'un graphe G et U , V deux sacs adjacents de *T*. Soient T_1 *et* T_2 les deux arbres obtenus en supprimant l'arête *(U , V)* dans *T*. Soient $V_1 = \bigcup_{X \in V(T_1)} X$ et $V_2 = \bigcup_{X \in V(T_2)} X$. Pour toute paire de sommets *(u , v)* telle que $u \in V_1$ *et* $v \in V_2$, toute chaîne dans G entre *u et v* utilise au moins un sommet de $U \cap V$.

Pour illustrer cette proposition, revenons à la figure 2.4 . On voit par exemple que si dans *T* on retire le sac {*b,c,e,f* } ainsi que les sommets *b,c,e,f* des autres sacs, alors on obtient trois arbres :

- un arbre avec un unique sac contenant le sommet *a*.
- un arbre avec deux sacs : {*i*} et {*i,h*},
- et un arbre avec deux sacs : {*d*} et {*d,g*},

Dans le graphe G il en est de même ; si on enlève les quatre sommets *b,c,e,f,* on obtient trois composantes connexes : une contenant le sommet *a*, une contenant les sommets *i,h,* et une autre contenant les sommets *d,g.*

2.3.2 Largeur arborescente

Définition 2.6

La largeur *d'une décomposition arborescente* est le nombre maximum de sommets contenus dans un de ses sacs moins 1.

$$\text{Largeur}(T) = \max_{B \in V(T)} \{ |B| - 1 \}$$

La largeur arborescente d'un graphe (en anglais Tree-width) est la plus petite des Largeurs de toutes les décompositions arborescentes possibles de G.

$$\text{tw}(G) = \min_{T} \{\text{Largeur}(T)\}$$

Dans la figure 2.4, la décomposition arborescente proposée est de largeur 3, car son plus gros sac contient 4 sommets de G. Notons que c'est d'ailleurs le mieux que l'on puisse faire. En effet il est possible de montrer qu'il n'existe pas de décomposition arborescente de ce graphe qui soit de largeur 1 ou 2 (ce graphe est de largeur minimum 3).

Autour de la largeur arborescente se sont développées une théorie et une littérature très volumineuses. En effet il a été constaté assez tôt que certains problèmes réputés difficiles sont résolubles pour des graphes de largeur arborescente petite. Beaucoup de problèmes classiques de l'algorithmique des graphes, qui sont NP-complets en général, peuvent être résolus en temps polynomial, voir linéaire, pour les graphes ayant une largeur arborescente bornée par une constante. On y retrouve par exemple le problème du stable maximum ou du cycle hamiltonien. Il existe des résultats qui identifient des classes de problèmes résolubles en temps linéaire pour des graphes de largeur arborescente bornée. Les travaux de Courcelle [29], Arnborg, Lagergren et Seese [3] et de Courcelle et Mosbah [30] ont aboutis à la conclusion que tous les problèmes exprimables par des formules monadiques étendus de

second ordre peuvent être résolus en temps linéaire . Schématiquement , les problèmes des graphes que l'on peut formuler à l'aide des opérateurs logiques ($\wedge,\vee,\neg,\Rightarrow$) , des quantificateurs sur des sommets, des arêtes, ensembles de sommets, et ensembles d'arêtes (comme $\exists$ W $\subseteq$ V , $\forall$ e $\in$ E) , de tests d'appartenance ou d'adjacence ($v \in$W , $xy \in$ E) et certaines extensions peuvent être résolus pour des graphes de largeur arborescente bornée, les extensions permettent de prendre en considération des problèmes d'optimisation , comme le problème du stable maximum.

Malheureusement, trouver une décomposition arborescente de largeur minimum est un problème NP-complets [2], et ce même dans les classes de graphes assez restreintes comme les graphes de degré borné [22], les graphes bipartis ou les graphes de co-comparabilité [45].

Il existe tout de même certaines familles de graphes pour lesquelles, il est possible de calculer la largeur arborescente. Par exemple le graphe complet est de largeur n-1, c'est le pire des cas. Les arbres eux, sont exactement les graphes de largeur arborescente 1, ceci est censé justifier le -1 dans la définition de la largeur arborescente.
La largeur arborescente d'un graphe peut se caractériser à l'aide des triangulations de la manière suivante :

Théorème 2.5 [46] *la largeur arborescente de G est égale au minimum, parmi toutes les triangulations possibles* G'*, de* $w(G') - 1$.

Le calcul de la largeur arborescente d'un graphe G peut donc se faire en cherchant des triangulations de G qui minimisent la taille de la clique maximum. On peut même se limiter à chercher parmi les triangulations minimales de G.

2.3.3 Séparateurs minimaux

Les algorithmes de calcul de la largeur arborescente passent tous par la notion de séparateur minimal.
On va d'abord les définir, ensuite on étudiera en détail les séparateurs minimaux des graphes triangulés et finalement, on va montrer comment énumérer tous les séparateurs minimaux d'un graphe.

Soit G = (V , E) un graphe quelconque. Un sous ensemble $S \subseteq V$ est un *séparateur* de G si et seulement si le graphe G[$V \setminus S$] n'est pas connexe. *S* est appelé un *ab-séparateur* si et seulement si *a* et *b* sont dans deux composantes connexes différentes de G[$V \setminus S$]. *S* est un *ab-séparateur minimal* si et seulement si *S* est un *ab*-séparateur et aucun sous ensemble de *S* n'est aussi un *ab-séparateur*. *S* est un *séparateur minimal* si et seulement si, il existe *a* et *b* tels que *S* soit un *ab*- séparateur minimal.

La figure 2.5 présente par exemple un graphe G et *S* un de ses séparateurs minimaux.

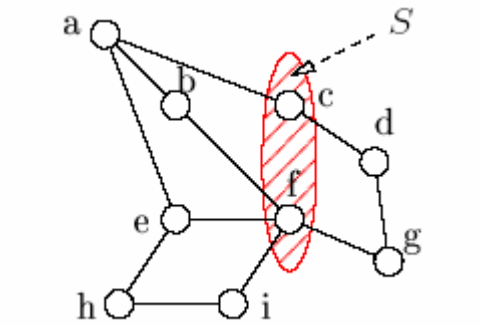


Figure 2.5 – Exemple de séparateur minimal.

Définition 2.7 Etant donné *S* un séparateur de G, une composante connexe C de G[$V \setminus S$] est appelée une composante connexe pleine si tout sommet de *S* a au moins un voisin dans C.
Autrement dit :
C est une composante connexe pleine si son voisinage contient *S*. Le voisinage d'un sous ensemble de sommets V' étant défini par :

$$N(V') = \left(\bigcup_{u \in V'} N(u) \right) \setminus V'$$

Proposition 2.2 [25] *S* est un séparateur minimal de G si et seulement si *S* a au moins deux composantes connexes pleines.

Dans le graphe de la figure 2.5 , le séparateur minimal *S* possède bien deux composantes connexes pleines : $\{d,g\}$ et $\{a,b,e,h,i\}$.
Il faut remarquer qu'un *ab*-séparateur *S* d'un graphe peut être vu comme un ensemble de sommets par lequel toute chaîne entre a et b doit passer. De plus *S* est minimal si et seulement si pour tout sommet *u* de *S*, il existe une chaîne entre *a* et *b* qui intersecte *S* uniquement en *u*.

Une nouvelle caractérisation des graphes triangulés peut être faite à l'aide des séparateurs minimaux. Cette caractérisation est due à Dirac en 1961 [32] :

Théorème 2.6 [32] *Un graphe est triangulé si et seulement si tous ses séparateurs minimaux sont des cliques.*

Ce théorème peut se reformuler en termes d'arbres de cliques.

Théorème 2.7 [46] *Soit G un graphe triangulé et T un arbre de cliques quelconque de G. Un ensemble de sommets S est un séparateur minimal de G si et seulement si il existe deux sacs B et B' adjacents dans T tels que $S = B \cap B'$.*

Il existe également un résultat concernant le diamètre des séparateurs minimaux dans les graphes de cordalité bornée :

Théorème 2.8 [41] *Tout séparateur minimal d'un graphe de cordalité bornée par k est de diamètre au plus k/2.*

2.3.4 Triangulations minimales et séparateurs minimaux

Dans ce paragraphe, nous allons voir la relation existante entre les séparateurs minimaux d'un graphe et ses triangulations minimales.

Proposition 2.3 [48] *Soit H une triangulation minimale d'un graphe G. Tout séparateur minimal S de H est aussi un séparateur minimal dans G. De plus les composantes connexes et les composantes connexes pleines induites par S sont identiques dans G[V \ S] et dans H[V \ S].*

Définition 2.8 *Soient S_1 et S_2 deux séparateurs de G. On dit que S_1 croise S_2 si S_1 intersecte au moins deux composantes connexes de G[V \ S_2]. Dans le cas contraire on dira que S_1 est parallèle à S_2.*

Dans sa thèse Parra [51] a montré que :

Lemme 2.1 *Les relations de parallélisme et de croisement sont symétriques entre séparateurs minimaux.*

Par exemple dans la figure 2.6 , les séparateurs minimaux S_1 et S_2 sont parallèles, de même pour S_1 et S_3. Par contre S_2 et S_3 se croisent.

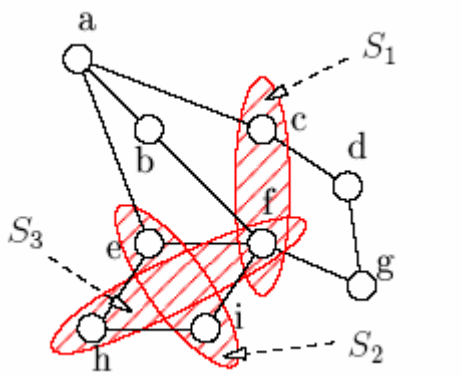


Figure 2.6 – Croisement et parallélisme entre séparateurs minimaux.

Dans leur étude sur la formation des triangulations minimales, Parra et Scheffler ont montré que :

Théorème 2.9 [52] *Toute triangulation minimale d'un graphe G peut être obtenue en prenant un ensemble maximale de séparateurs minimaux deux à deux parallèles et en les complétant en cliques.*

2.4 Largeur arborescente des graphes faiblement triangulés

Dans cette partie, nous allons voir comment calculer une décomposition arborescente de largeur minimum pour tout graphe faiblement triangulé. Ces travaux sont du à Bouchitté et Todinca [25] .

2.4.1 Calcul de la largeur arborescente d'un graphe G

Pour calculer la largeur arborescente d'un graphe G, il faut maîtriser les cliques qui apparaissent dans les triangulations minimales de G , c'est-à-dire les cliques maximales potentielles de G .

Définition 2.9 Un ensemble de sommets Ω d'un graphe G est appelé clique maximale potentielle s'il existe une triangulation minimale H de G telle que Ω soit une clique maximale de H.

On peut caractériser les cliques maximales potentielles Ω simplement en fonction des composantes connexes de G \ Ω .

Soit K un ensemble de sommets d'un graphe G . On note $C_1(K)$, …, $C_p(K)$ les composantes connexes de G \ K .

Pour chaque $1 \leq i \leq p$, on pose :

$S_i(K)$: ensemble de sommets de K ayant au moins un voisin dans $C_i(K)$

On a que si $S_i(K) = K$, on dit que $C_i(K)$ est une composante connexe pleine associée à K.

Soit K un ensemble de sommets du graphe G . On peut vérifier en un temps $O(n^3)$ si K est une clique maximale potentielle.

Algorithme de reconnaissance des cliques maximales potentielles

Entrée : le graphe G= (V, E) et l'ensemble de sommets $K \subseteq V$

Sortie : Booléen indiquant si K est une clique maximale potentielle

- Calculer les composantes connexes C_i ($1 \leq i \leq p$) de G \ K
- Calculer les ensembles S_i
- Si il existe un i tel que : $S_i = K$ alors

  Retourner « K n'est pas une clique potentielle »

  Pour $i = 1$ à p

Compléter S_i

Pour chaque couple de sommets $x , y \in K$

Si x et y ne sont pas adjacents dans $G_{\{S1 , ... , Sp\}}$ alors

Retourner « K n'est pas une clique potentielle »

Retourner K est une clique maximale potentielle

D'autre part, on dispose d'un algorithme pour le calcul de la largeur arborescente d'un graphe en temps polynomial lorsque le nombre des cliques maximales potentielles est polynomial en la taille du graphe .Cet algorithme fonctionne en deux temps, d'abord, il vérifie si $tw(G) \leq k$ et ensuite , il donne une décomposition arborescente de G de largeur au plus k .

Définitions et Notations 2.10

Soit S un ensemble de sommets (pas forcément un séparateur)

On note $C_G(S)$: l'ensemble des composantes connexes de $G \setminus S$.

Soit $C \in C_G(S)$;

Si tout sommet de S a au moins un voisin dans C, on dira que C est une composante connexe pleine associée à S.

Soit S un séparateur minimal de G et $C \in C_G(S)$ une composante connexe de $G \setminus S$ .On dit que $(S, C) = S \cup C$ est un bloc de S dans G.

Si C est une composante connexe pleine associée à S, on dira que le bloc (S , C) est un bloc plein.

Soit $\Im = \{\Omega_1, ... , \Omega_p\}$ une famille de cliques maximales potentielles de G.

$\Gamma_\Im = \{S / \exists \Omega \in \Im$ tel que $S \subset \Omega\}$ famille de séparateurs minimaux de G contenus dans au moins une de ces cliques potentielles.

On dira que $\Im$ est une famille de cliques maximales potentielles complète si pour tout séparateur minimal $S \in \Gamma_\Im$ et pour tout bloc plein (S , C) de S , il existe une clique maximale potentielle $\Omega \in \Im$ telle que ;

$$S \subset \Omega \subseteq (S , C).$$

On a le théorème suivant :

Théorème2.10 [25] Soit G un graphe, et $\Im$ une famille complète de cliques maximales potentielles de G. Si $\Im$ n'est pas vide, on peut construire une triangulation minimale H de G telle que toute clique maximale potentielle de H soit un élément de la famille $\Im$.

On a l'algorithme suivant :

Description de l'algorithme

On commence par calculer la plus grande famille complète $\Im = \{\Omega_1 , \ldots . , \Omega_p\}$ de cliques maximales potentielles de G telle que toute clique potentielle $\Omega \in \Im$ ait au plus k + 1 sommets. Il s'agit d'un processus d'élimination, on veut repérer toutes les cliques potentielles maximales qui n'entrent pas dans la composition d'un arbre de cliques de largeur k.

Algorithme d'élimination

Entrée : G , ses séparateurs minimaux et ses cliques maximales potentielles ;
l'entier k.

Sortie : la plus grande famille complète $\Im$ de cliques maximales potentielles,
dont tous les éléments ont au plus k + 1 sommets

1. Calculer toute les cliques maximales potentielles de G ayant k + 1 sommets . Soit $\Im$ la famille de ces cliques potentielles et soit $\Gamma_{\Im}$ l'ensemble des séparateurs minimaux contenus dans ces cliques
2. Extraire de $\Im$ la plus grande sous famille complète de cliques potentielles :

Tant que ; il existe un $S \in \Gamma_{\Im}$ et un bloc (S,C) de S tels qu'aucune clique potentielle $\Omega \in \Im$ ne satisfait $S \subset \Omega \subseteq (S,C)$

faire

- $\Im = \Im \setminus \{\Omega / S \subset \Omega\}$
- mettre à jour $\Gamma_{\Im} = \{S / \exists \Omega \in \Im$ tel que $S \subset \Omega\}$

3. Retourner $\Im$

Algorithme d'extraction

Entrée : G et une famille complète non vide $\Im$ de cliques maximales
Potentielles.

Sortie : un arbre de cliques *T* d'une triangulation minimale H de G telle que

toutes les cliques maximales de H soient des éléments de $\mathfrak{I}$

1. Etape d'initialisation

- choisir une clique maximale potentielle $\Omega \in \mathfrak{I}$
- créer la racine de τ et l'étiqueter par Ω
- marquer la racine comme « traitée »
- pour tout bloc plein (S,C) avec $S \subseteq \Omega$ et $\Omega \not\subset (S,C)$;
    - créer une feuille liée à la racine
    - étiqueter la famille par (S,C) et l'arête adjacente par S
    - marquer la feuille « non traitée »
- mettre $\mathfrak{I}' = \{\Omega\}$

2. Etape d'expansion

Tant qu'il existe un nœud N non traité faire

- Soit (S,C) l'étiquette de N .Choisir une clique maximale potentielle $\Omega \in \mathfrak{I}$ telle que $S \subset \Omega \subseteq (S,C)$
- Etiqueter le nœud N par Ω
- Pour tout bloc plein (S', C') tel que $S' \subset \Omega$, $S' \not\subset S$, $\Omega \not\subset (S',C')$
    - créer une feuille N' adjacente à N
    - étiqueter N' par (S',C')
    - étiqueter l'arête NN' par S'
    - marquer N' comme non traité
- Mettre $\mathfrak{I}' = \mathfrak{I}' \cup \{\Omega\}$
- Marquer le nœud N comme traité

3. Retourner $H = G\Gamma_{\mathfrak{I}'}$ et l'arbre étiqueté *T*.

En partant d'une famille complète de cliques maximales potentielles, on construit directement un arbre (I , E_T) par l'algorithme d'extraction qui sera à la fin , un arbre de H et dont les nœuds (ou sacs) seront étiquetés par des éléments de la famille , de plus cet arbre est de largeur arborescente minimale .

2.4.2 Calcul de la largeur arborescente des graphes faiblement triangulés

Définition 2.11 Un graphe G est dit faiblement triangulé si , à la fois dans G et dans son complémentaire $\overline{G}$, tout cycle ayant au moins cinq sommets possède une corde.

Les graphes faiblement triangulés contiennent :

- les graphes triangulés : les graphes dont tout cycle de longueur ≥ 3 possède une corde .
- les graphes bipartis – chordaux : les graphe bipartis ,sans cycles induits de longueur au moins 6 .
- les graphes HHD-free : les graphes sans cycles induits de longueur au moins 6 et qui ne possèdent pas de maisons , de trous et de dominos comme sous graphes induits.
- les graphes à distance héréditaire : les graphes qui préservent les distances dans tous les sous graphes connexes induits

Définition 2.12 Une paire de sommet $\{x, y\}$ est une 2 – paire dans G ssi toute chaîne sans corde les reliant est de distance 2.

On a les résultats suivants :

Proposition 2.4 [25] Soit G = (V,E) un graphe faiblement triangulé non complet et soit $\{x,y\}$ une deux paire de G .Considérons une clique maximale potentielle Ω de G telle que *x* et *y* soient dans des composantes connexes distinctes de G \ Ω , alors Ω est une clique maximale potentielle du graphe $G^{/} = (V, E \cup \{xy\})$.

Corollaire 2.1 [25] Soit Ω une clique maximale potentielle du graphe faiblement triangulé G=(V,E). Soient *x* et *y* deux sommets formant une deux paire de G et $G^{/} = (V, E \cup \{xy\})$.
Deux situations sont possibles :

- Ω s'écrit sous la forme $S_{xy} \cup \{x\}$ ou $S_{xy} \cup \{y\}$ avec $S_{xy} = N(x) \cap N(y)$
- Ω est une clique maximale potentielle de $G^{/}$.

Corollaire 2.2 [25] Un graphe faiblement triangulé a au plus $2\overline{m}$ + 1 cliques maximales potentielles ($\overline{m}$ = nombre d'arêtes de $\overline{G}$).

Corollaire 2.3 [25] Les clique maximales potentielles d'un graphe faiblement triangulé peuvent être énumérés en temps polynomial en $O(n^3\overline{m})$.

Algorithme : Cliques maximales Potentielles des graphes faiblement triangulés

Entrée : le graphe faiblement triangulé G_0

Sortie : Ses séparateurs minimaux et ses cliques maximales potentielles

Pour $i = 1$ à $\overline{m}$

Calculer une deux paire x_i , y_i de G_{i-1}

$S_{x_i y_i} \leftarrow N_{Gi\text{-}1}(x_i) \cap N_{Gi\text{-}1}(y_i)$

$G_i \leftarrow (V , E(G) \cup \{x_i y_i\})$

Vérifier si $S_{x_i y_i}$ est un séparateur minimal de G_0

Vérifier si $S_{x_i y_i} \cup \{x_i\}$ ou $S_{x_i y_i} \cup \{y_i\}$ sont des cliques maximales potentielles de G_0

2.5 Décompositions arborescentes réduites

Nous venons de définir de manière générale la notion de décomposition arborescente, qui sera à la base des schémas de routage.
Nous allons à présent voir qu'il est possible de considérer un seul type de décomposition arborescente : les décompositions arborescentes réduites.

Définition 2.13 Une décomposition arborescente T est dite réduite, s'il n'existe pas de sac contenu dans un autre. C'est-à-dire que pour tout $B_1 \in V(T)$ et tout $B_2 \in V(T)$, on a $B_1 / B_2 \neq \emptyset$ et $B_2 / B_1 \neq \emptyset$.($V(T)$ étant l'ensemble de sacs de T)

Proposition 2.5 [33] Pour toute décomposition arborescente T , il est possible de rendre T réduite sans modifier sa largeur.

Preuve : Soit T une décomposition arborescente supposée non réduite, c'est-à-dire qu'il existe deux sacs B_1 et B_2 tels que $B_1 \subseteq B_2$. De par la règle 3 de la définition 2.5 d'une décomposition arborescente, B_1 est également contenu dans tous les sacs situés entre B_1 et B_2. On peut donc supposer que B_1 et B_2 sont voisins dans T.
Or il est claire qu'en contractant l'arête (B_1 , B_2), telle que représentée dans la figure 2.7 , T reste une décomposition arborescente de G.

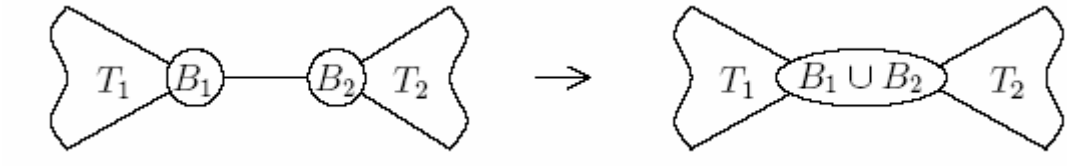


Figure 2.7 – Contraction d'une arête d'une décomposition arborescente.

De plus comme $B_1 \subseteq B_2$, il est clair qu'après cette contraction, la largeur de T n'a pas changé.
On obtient donc une décomposition arborescente qui est plus réduite : le nombre des sacs inclus les uns dans les autres a strictement diminué. De plus la largeur de T n'a pas changé.
Pas à pas, on finit par réduire totalement T sans jamais modifier sa largeur.

La proposition précédente montre bien que, sans perte de généralité, nous pourrons supposer qu'une décomposition arborescente est toujours réduite. Ceci est d'autant plus intéressant que, comme le montre la proposition suivante, les décompositions arborescentes réduites ont un nombre de sacs moins important que le nombre de sommets du graphe. Ceci n'a rien d'évident pour une décomposition arborescente en général.

Proposition 2.6 [19] *Soit T une décomposition arborescente réduite d'un graphe G à* $n \geq 2$ *sommets, alors* $|V(T)| \leq n - 1$.

CHAPITRE 3

Problèmes de routage

Dans ce chapitre, nous donnons les principaux résultats sur le routage dans les classes de graphes ; les arbres, les graphes triangulés, et les graphes qui admettent une décomposition arborescente de faible diamètre.

3.1 Le problème de routage dans les réseaux

Définition 3.1

Le routage est une méthode d'acheminement des informations à la bonne destination à travers un réseau de connexion donné. Le problème de routage consiste pour un réseau dont les arêtes, les nœuds et les capacités sur les arêtes sont fixés, à déterminer un acheminement optimal des paquets (de messages, de produits …) à travers le réseau au sens d'un certain critère de performance.

Exemple

Si on suppose que les coûts des liens sont identiques le chemin indiqué dans la figure suivante est le chemin optimal reliant la station source et la station destination. Une bonne stratégie de routage utilise les plus courts chemins dans le transfert des données entre les deux stations.

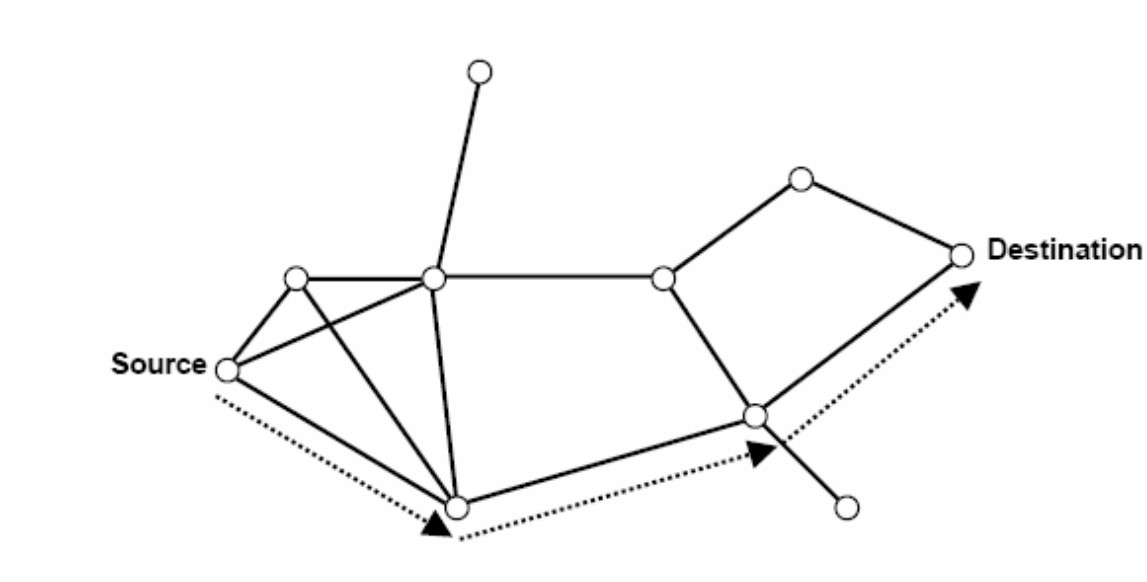


Figure 3.1 - Le chemin utilisé dans le routage entre la source et la destination.

3.1.1 Routeur et schéma de routage

Pour des raisons évidentes liées à la réalisation matérielle du réseau, toutes les machines ne sont pas reliées entre elles. Aussi, pour que deux machines puissent communiquer, il est souvent indispensable que l'information transite par des machines intermédiaires dont le seul rôle est de faire suivre le message. Pour effectuer cette opération, on dote de manière standard chaque machine d'un routeur. Un routeur est une unité indépendante à laquelle sont rattachés les liens de communication et dont le but est de déterminer une route pour un message lui parvenant. On dira qu'un message est ainsi routé à l'intérieur du réseau.

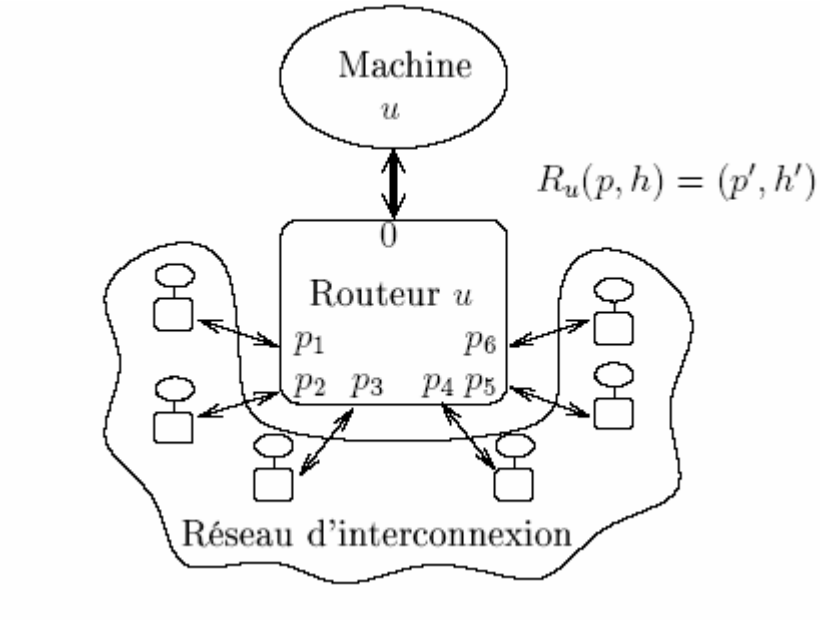


Figure 3.2 – Un exemple de routeur

Schémas de routage

Un schéma de routage est composé de :

1. *adresse (u)* : adresse de la machine *u* (chaîne binaire).
2. *les en-têtes ajoutées à tout message* : est constituée généralement de l'adresse de la destination.
3. *un numéro de port* : entier positif qui identifie tout lien de communication.
   Une arête est décomposée en deux brins (deux demi- arêtes), on note port (*u, v*) le numéro de port du brin allant de *u* à *v*. Il n'y a pas de liens de correspondance entre port (*u, v*) et port (*v, u*).
   Le numéro 0 est attribué au lien faisant connexion entre le routeur et la machine qui lui est associée.
4. *fonction locale de routage* : notée R_u , permet lorsque un message arrive par un port *p* avec un en-tête *h* de calculer un nouvel en-tête h' et port de sortie p'. Le message sera ainsi envoyé par le port p' avec l'en-tête h'.
5. *la mémoire locale de chaque routeur* : là est stockée sa fonction locale et possède ainsi un certain nombre d'information sur la topologie du réseau. ces informations vont intervenir dans le calcul de la fonction locale de routage.

Un schéma de routage est dit *valide* si on est capable d'envoyer un message entre tout couple de sommets, c'est–à–dire, tout sommet *u* est capable en utilisant sa mémoire locale, et l'adresse de n'importe quel autre sommet *v,* de construire un en-tête de message permettant de faire circuler un message dans le réseau entre *u* et *v* (sans faire de boucles).

3.2 Caractéristiques d'un schéma de routage

3.2.1 Taille des informations

C'est la quantité des informations nécessaires au réseau, et pour la mesurer nous avons besoins de deux critères essentiels :

- la mémoire locale du sommet *u* notée *mem(u).*
- l'adresse du sommet *u* (car on y stocke la route permettant d'y arriver depuis n'importe quel autre sommet) on note *adresse (u)*.

Donc, il faut pour des raisons évidentes de coût, minimiser cette quantité d'information. ie ; Optimiser la complexité mémoire de nos schémas de routages.
$|adresse\ (u)| + |mem(u)|$ = nombre de bits nécessaires pour coder respectivement l'adresse de la machine *u* et sa mémoire locale.

3.2.2 Longueur des routes

Etant donné un schéma de routage dans un graphe G, la longueur de la route produite par la fonction de routage *R* entre deux sommets *u* et *v*, notée $\rho_R(u,v)$, est le nombre d'arêtes traversée par le message depuis la source avant d'atteindre la destination.

Un schéma de routage est dit de plus court chemin si pour tout couple de sommets (u,v) on a : $\rho_R(u,v) = dist_G(u,v)$.

Un facteur d'étirement d'un schéma de routage est le plus petit réel *a* tel que pour tout couple (u , v) de sommets $\rho_R(u,v) \leq a.\ dist_G(u,v)$
La déviation du schéma de routage est le plus petit entier δ tel que

$$\rho_R(u,v) \leq dist_G(u,v) + \delta .$$

La dilatation du schéma de routage est la largeur de la plus grande route dans G :

$$max_{u,v}\ \{\rho_R(u,v)\}.$$

En général, il faut trouver un compromis entre la taille de la mémoire locale des sommets, et la longueur des routes, plus l'information stockée sur un sommet est faible, et plus le sommet a tendance à s'écarter des plus courts chemins lorsqu'il fait suivre un message.

3.2.3 Rapidité de décision

C'est le temps nécessaire à un routeur pour décider vers où il doit faire transiter le message. Ce paramètre se calcule en étudiant la complexité en temps de la fonction R_u.

3.3 Méthodes standard pour implémenter un schéma de routage

On a deux méthodes : routage par intervalles et tables de routage (qui nous intéresse)

3.3.1 Tables de routage

Le routage par table de routage est une méthode assez classique car elle permet d'implémenter n'importe quelle fonction de routage sur n'importe quel réseau.

Dans cette solution, la fonction d'adressage des sommets est une bijection de V(G) dans $\{1,2,\ldots,n\}$. Les en-têtes se limitent à l'adresse de la destination. Ainsi, en-tête et adresses sont de taille $\lceil \log n \rceil$ bits. La mémoire locale d'un sommet se compose d'une table à n entrées, la $i^{ème}$ entrée contenant le numéro du port à utiliser pour faire transiter un message destiné au sommet numéro i. La mémoire locale d'un sommet u est donc de taille

$O(n \log d)$ bits , avec $d = deg(u)$.

Par exemple la figure 3.3 suivante représente un graphe à 5 sommets dotés d'un schéma de routage par table de routage. Ainsi lorsque le sommet 2 veut envoyer un message au sommet 5, il utilise le port numéro 1, ce qui le conduit au sommet 1 ou il est redirigé vers le port numéro 2 pour arriver en 5.

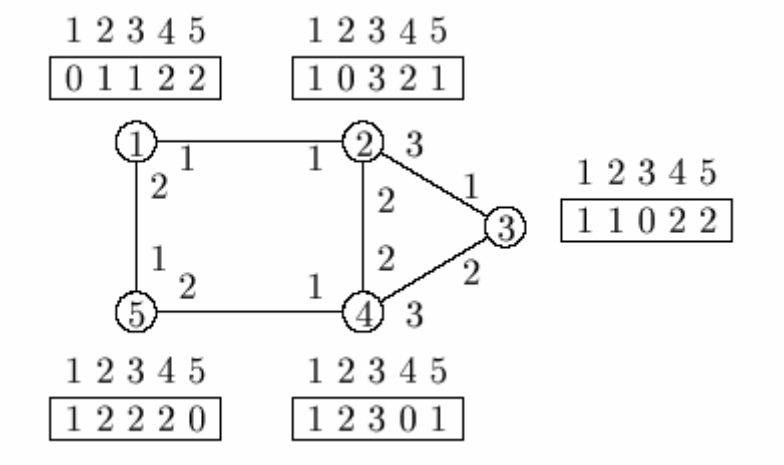


Figure 3.3 – Table de routage.

3.3.2 Routage par intervalles

On vient de voir que toute fonction de routage peut être réalisée par des tables de routage. Cependant pour réduire la mémoire locale nécessaire sur chaque sommet, Khatlib et Santoro ont introduit dans [47] une autre méthode pour coder les tables de routage : Le routage par intervalles. Ils proposent de regrouper les destinations empruntant le même port de sortie en intervalles. La mémoire locale d'un sommet *u* est alors une table avec *deg(u)* entrées, l'entrée numéro *i* indiquant l'ensemble des destinations (une liste d'intervalles) que l'on peut atteindre en utilisant le port de sortie numéro *i*.

Un des paramètres importants dans les schémas de routage par intervalles est la compacité c'est-à-dire le nombre maximum d'intervalles sur un lien de communication. En effet la mémoire locale d'un sommet est *O(cd log n)* ou *c* est la compacité et *d* le degré maximum.

Un exemple de routage par intervalles de compacité 1 est donné par la figure 3.4 . Dans cet exemple lorsque le sommet numéro 3 veut envoyer un message au sommet numéro 5, il le fait suivre au sommet 4 qui lui-même le fera suivre au sommet 5.

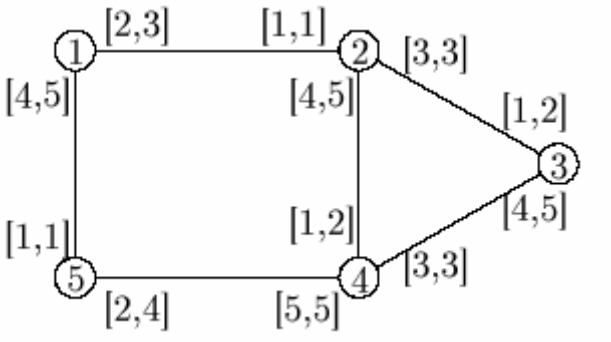


Figure 3.4 – Routage par intervalle

3.4 Routage compact

Le routage compact est la méthode générale pour tenter d'optimiser la taille des adresses et de la mémoire locale de chaque sommet tout en garantissant des routes pas trop longues.

3.4.1 Routage compact dans les arbres

Les schémas de routage selon les plus courts chemins dans les arbres sont du à Fraigniaud et Gavoille dans [38] et à Thorup et Zwick dans [58].

Théorème 3.1 [38] Tout arbre à n nœuds admet un schéma de routage de plus courts chemins constructibles en temps polynomial, dont la fonction de routage est calculable en temps constant et tel que les adresses et les mémoires locales sont de tailles :
- $O(\log^2 n / \log \log n)$ bits si l'on ne peut pas choisir la numérotation des ports.
- $\log n + O(\log n / \log \log n)$ bits sinon.

3.4.2 Problème de routage dans les graphes triangulés

Les schémas de routage dans les graphes triangulés sont du aux travaux de Dourisboure et Gavoille dans [33], [34], [35].

a) Routage dans les graphes triangulés de clique maximum bornée

Soit G un graphe triangulé de clique maximum k, c'est-à-dire que $\omega(G) = k$. T un arbre de cliques de G enraciné, et S un arbre couvrant G par des plus courts chemins enraciné en un sommet r appartenant à la racine de T.

Exemple :

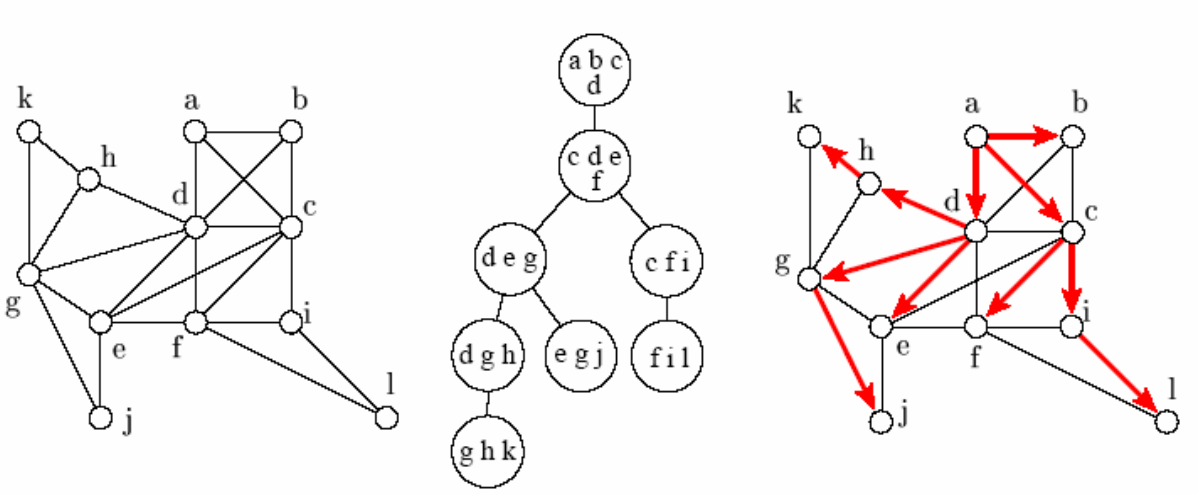


Figure 3.5 – Un graphe triangulé G, un arbre de cliques *T* et un arbre S couvrant G.

Avec ces deux arbres *T* et S, on va pouvoir définir une stratégie de routage dans le graphe G. Pour envoyer un message d'un sommet *u* à un sommet *v*, on commence par faire « remonter » le message dans l'arbre *T* jusqu'à arriver dans un sac qui est un ancêtre de B(*v*), (B(*v*) : est le sac du sommet *v*) , puis on le fait « descendre » dans S jusqu'à *v*. Le schéma basé sur cette stratégie est bien de déviation 2. Par contre, il a une complexité mémoire qui dépend de la clique maximum car au moment de changer d'arbre, il faut que le sommet connaisse tous les sommets qui sont dans le même sac que lui.

Proposition 3.1 [33] Soient *u* et *v* deux sommets adjacents de G. Au moins une des deux affirmations suivantes est vraie.

1. *B(u)* est un ancêtre de *B(v)* dans *T* et $u \in B(v)$.
2. *B(v)* est un ancêtre de *B(u)* dans *T* et $v \in B(u)$.

Proposition 3.2 [33] Soient *u* et *v* deux sommets de G tels que *B(u)* est un ancêtre de *B(v)* dans *T* . Il existe au moins un sommet $w \in B(u)$, et au plus deux, tel que *w* est un ancêtre de *v* dans *S* .

On aboutit au théorème suivant :

Théorème 3.2 [34] Si on suppose que l'on peut choisir la numérotation des ports, alors tout graphe triangulé de clique maximum k admet un schéma de routage de déviation 2 sans boucle, avec des adresses de taille $(2 + o(1)) \log n$ bits et des mémoires locales de taille $O(k \log n)$ bits. De plus la mise en place de ce schéma se fait en temps polynomial et la fonction de routage est calculable en temps $O(\log k)$.

b) Routage compact dans les graphes triangulés quelconques

Dans ce schéma, on a besoin de construire un arbre de cliques T de G, et de plusieurs arbres couvrant G ; un par sac de T.
Etant donné qu'un sac X de T, l'arbre couvrant G associé à X est noté S_X et sa racine sera un sommet $r_X \in X$ et tel que $B(r_X) = X$.
La stratégie de routage utilisée pour envoyer un message depuis un sommet u vers un sommet v, est de suivre la route entre u et v dans un arbre S_X tel que le sac X soit situé sur le chemin dans T entre $B(u)$ et $B(v)$. Ainsi, la déviation est au plus 2.
Mais de tels sacs peuvent être très nombreux. Aussi, pour limiter la mémoire locale des sommets, on utilise un arbre hiérarchique $\mathcal{H}$ de T. Cet arbre de profondeur au plus $\log n$, permettra à un sommet de ne conserver que l'information de routage relative aux arbres S_X pour lesquels X est un ancêtre dans $\mathcal{H}$ de $B(u)$, ainsi que celle de l'arbre $S_{B(u)}$. Ainsi pour envoyer un message à un sommet v, on choisira d'utiliser l'arbre S_X où $X = nca_{\mathcal{H}}(B(u), B(v))$ avec ($nca_{\mathcal{H}}(B(u), B(v)$ signifie : le plus petit ancêtre commun entre $B(u)$ et $B(v)$ dans l'arbre hiérarchique $\mathcal{H}$).

Définition 3.2 Il est connu que tout arbre à n nœuds possède un médian, c'est-à-dire un sommet s tel que $T \setminus \{s\}$ est une foret dont chaque arbre possède au plus $n/2$ nœuds. Un arbre hiérarchique de T est un arbre enraciné H défini de manière récursive par : la racine de H est un médian de T, et ses enfants sont les racines des arbres hiérarchiques des arbres de $T \setminus \{s\}$. Notons que H et T ont le même ensemble de nœuds mais H est de profondeur au plus $\log n$.

Exemple

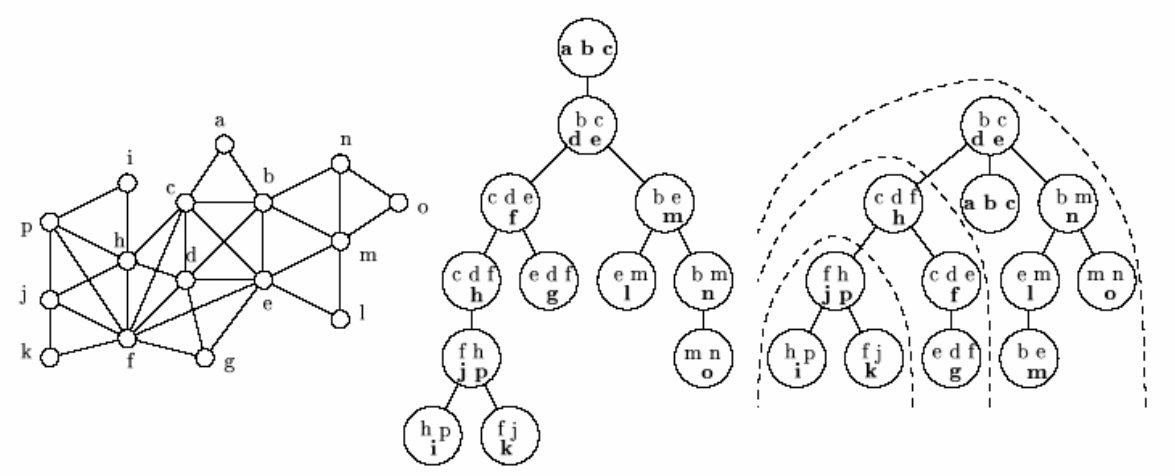


Figure 3.6 – Un graphe triangulé G, un arbre de cliques T et un arbre hiérarchique $\mathcal{H}$.

On aboutit au théorème suivant :

Théorème 3.3 [34] Tout graphe triangulé admet un schéma de routage de déviation 2, sans boucles, in dépendant de la numérotation des ports, utilisant des adresses et des mémoires locales de taille $O(\log^3 n / \log \log n)$ bits et des en-têtes de message de taille $O(\log^2 n / \log \log n)$ bits. De plus une fois calculés par la source du message, les en-têtes ne sont plus jamais modifiés. Ce schéma de routage peut être construit en temps polynomial et la fonction de routage s'exécute en temps constant.

c) Schéma de routage de déviation 1

Yon Dourisboure propose un schéma de routage de déviation 1, qui reprend le schéma de routage de Peleg [53] qui route selon les plus courts chemins, mais en compressant la taille des adresses et en s'autorisant une déviation de 1.

Théorème 3.4 [35] Tout graphe triangulé de clique maximum k admet un schéma de routage de déviation 1, sans boucle, tel que la mémoire locale d'un sommet est de taille $O(k \log^2 n + \log^3 n / \log \log n)$ bits. L'adresse d'un sommet est de taille $O(\log^3 n / \log \log n)$ bits et l'en-tête d'un message est de taille $O(\log^2 n / \log \log n)$ bits. De plus ce schéma peut être construit en temps polynomial.

3.5 Schéma de routage dans les graphes admettant une décomposition arborescente dont les sacs sont de faible diamètre

Yon Dourisboure et cyril Gavoille ont adapté un schéma aux graphes qui n'admettent un arbre de cliques, mais une décomposition arborescente dont les nœuds (les sacs) sont de diamètre au plus δ.

Le schéma de routage dans les graphes admettant une décomposition arborescente dont les sacs sont de diamètre borné, est très semblable à celui pour les graphes triangulés quelconques.

En effet les mêmes notions d'arbre hiérarchique $\mathcal{H}$, d'arbre couvrant G par des plus courts chemins S_X enraciné en r_X sont utilisées. Et la route produite entre u et v sera également la route dans l'arbre S_X où $X = \text{nca}_{\mathcal{H}}(B(u), B(v))$.

La différence réside dans le fait que T n'est plus un arbre de cliques de G, mais une décomposition arborescente de G. Les sacs ne sont plus de diamètre 1 mais au plus δ. On supposera sans perte de généralité que T est réduite.

On aboutit au résultat suivant :

Théorème 3.5 [3] Tout graphe admettant une décomposition arborescente dont les sacs sont de diamètre au plus δ admet un schéma de routage de déviation 2δ, sans boucles, et indépendant de la numérotation des ports. Dans ce schéma, les adresses et les mémoires locales sont de taille $O(\delta \log^3 n)$ bits par sommet.

CHAPITRE 4

La longueur arborescente

Les travaux présentés dans le chapitre 3 ont aboutit à la construction de schémas de routage compact de faible déviation dans les graphes admettant une décomposition arborescente dont les sacs sont de diamètre borné.

Cela a conduit à faire apparaître un nouvel invariant des graphes, la longueur arborescente qui est exactement le diamètre des sacs de la décomposition arborescente.

Dans ce chapitre, nous allons donner la définition de la longueur arborescente, ainsi que tous les résultats obtenus lors du calcul de cet invariant. Notamment, la longueur arborescente de certaines classes de graphes, comme les graphes planaires extérieures, et aussi des algorithmes d'approximations, qui permettent de calculer des décompositions arborescentes pour n'importe quel graphe.

4.1 Longueur arborescente

Définition 4.1 Le diamètre d'un sac B est la distance dans le graphe entre les deux sommets les plus éloignés du sac. $diam_G\ B = Max_{\ u,v\in B}\ \{dist_G\ (u\ ,\ v)\}$.

La longueur d'une décomposition arborescente *T* est le plus grand diamètre de ses sacs ;

$$longueur(\ T\ ) = Max_{\ B\in V(T)}\ \{diam_G(B)\}.$$

La longueur arborescente d'un graphe G (en anglais : tree-length) notée tl(G) est la plus petite des longueurs de toutes les décompositions arborescentes possibles de G :

$$tl(G) = Min_T\ \{longueur(T)\}.$$

Cette notion de longueur arborescente vient compléter celle de largeur arborescente qui a été abondamment étudiée

4.1.1 Applications de la longueur arborescente

Le paramètre « longueur arborescente » peut nous aider à aborder différemment certains problèmes ;

a) Construction d'étiquetages de distance

L'étiquetage de distance dans graphe G est un problème classique en théorie des graphes. Il consiste à trouver :

- un étiquetage des sommets de G, c'est-à-dire une fonction *L*, définie sur l'ensemble des sommets de G et qui à un sommet *u* , associe son étiquette *L(u)*.

- un décodeur de distances, c'est-à-dire une fonction *f* qui, étant donné deux étiquettes *L(u)* et *L(v)*, renvoie la distance entre *u* et *v* dans le graphe G.

Un étiquetage de distance est une (*a,b*)-approximation si pour tout couple de sommets (*u,v*) on a : $dist_G(u,v) \le f(L(u),L(v)) \le a\ .dist_G(u,v) + b.$

L'objectif est de trouver une (*a,b*)-approximation des distances avec des étiquettes de taille minimum et un décodeur le plus rapide possible.

On a le théorème suivant ;

Théorème 4.1 [47] Tout graphe de longueur arborescente δ admet un étiquetage de distance qui est une (1,δ)-approximation avec des étiquettes de taille O(log *n* log diam(G)) bits et dont la fonction de décodage est calculable en temps constant.

b) Recherche de sous graphes couvrant qui approximent les distances

Un sous graphe H couvrant un graphe G est (*a*,*b*)-approximant (en anglais (*a*,*b*)-spanner) si et seulement si pour toute paire de sommets (*u*,*v*) de G , $dist_H(u,v) \leq a \,.dist_G(u,v) + b$.
L'objectif est de trouver un sous graphe couvrant (*a*,*b*)-approximant avec un nombre minimum d'arêtes .

On a le théorème suivant ;

Théorème 4.2 [28] Soit G un graphe de longueur arborescente δ . G possède un sous graphe couvrant (1,2δ)-approximant avec O(δ . *n* log *n*) arêtes.

c) Mise au point de schémas de routage

Pour tout graphe de longueur arborescente δ , il est possible de faire des schémas de routage.
On a le théorème suivant ;

Théorème 4.3 [36] Tout graphe de longueur arborescente δ admet un schéma de routage de déviation 2δ et dont les adresses et les mémoires locales sont de tailles O ($\delta \log^3 n$) bits par sommet.

4.2 Propriétés relatives aux sacs d'une décomposition arborescente

On va présenter quelques outils généraux pour aider au calcul de la longueur arborescente.
La propriété suivante est une généralisation de la règle 3 de la définition 2.5 d'une décomposition arborescente :

Proposition 4.1 [36] Soit *T* une décomposition arborescente d'un graphe G et $V^{/}$ un sous ensemble de sommets de G. Si $V^{/}$ induit un sous graphe connexe de G, alors l'ensemble des sacs contenant au moins un sommet de $V^{/}$ forme un sous arbre.

Proposition 4.2 [36] Soient A et B deux sous ensembles de sommets d'un graphe G, tels que A et B induisent des sous ensembles connexes de G. Soit *T* une décomposition arborescente quelconque de G. L'une de ces deux affirmations est vraie :
- il existe un sac contenant un élément de A et un élément de B.
- il existe une arête $(X , Y) \in E(T)$ telle que $X \cap Y$ sépare A et B.

4.3 Exemples de calcul de la longueur arborescente

4.3.1 Longueur arborescente des graphes planaires extérieurs

Un graphe planaire est un graphe qu'il est possible de dessiner sur le plan sans que deux arêtes ne se croisent. Dans un dessin planaire, les arêtes partitionnent le plan en régions connexes appelées faces. De plus une de ces régions est non bornée, elle est appelée face extérieure, les autres sont appelées des faces internes.
Un graphe planaire est planaire extérieure s'il admet un dessin planaire tel que tous les sommets appartiennent à la frontière de la face extérieure.
Par exemple, le cycle est un graphe planaire extérieur. Le graphe présenté figure 4.1 l'est également.

Remarque
$\lceil .. \rceil$: désigne la partie entière par excès.

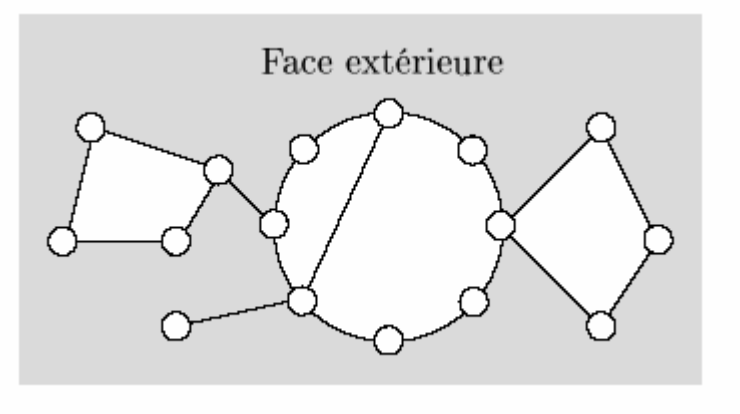


Figure 4.1 – Un exemple de graphe planaire extérieur.

Lemme 4.1 [36] Tout cycle de longueur k est de longueur arborescente $\lceil k/3 \rceil$ et de largeur arborescente 2.

Théorème 4.4 [36] Tout graphe planaire extérieur de cordalité k est de longueur arborescente $\lceil k/3 \rceil$. De plus il est possible d'obtenir en temps linéaire une décomposition arborescente de longueur $\lceil k/3 \rceil$ et de largeur 2.

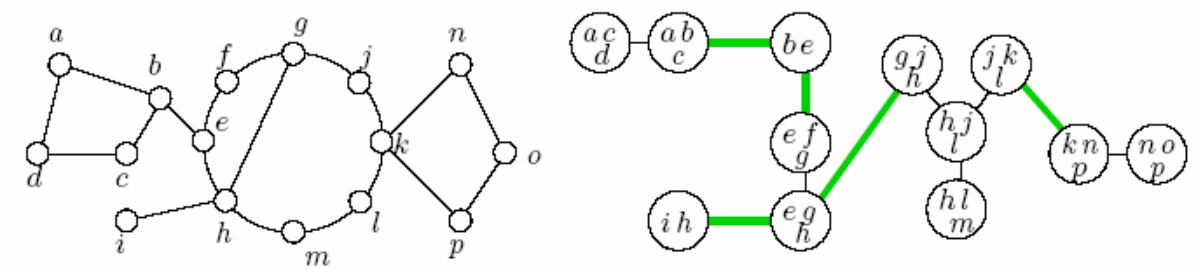


Figure 4.2 – Décomposition arborescente de longueur $\lceil k/3 \rceil$ et de largeur 2 d'un graphe planaire extérieur.

4.3.2 Longueur arborescente de la grille $p \times q$

La grille est un graphe très largement étudié car il représente naturellement l'architecture des machines parallèles.

Définition 4.2 La grille à p lignes et q colonnes, notée $M_{p,q}$, est le graphe défini par :

- $V(M_{p,q}) = \{(i , j) / i \in \{1,\ldots,p\}$ et $j \in \{1,\ldots,q\}\}$.

- $E(M_{p,q}) = \{((x , y) , (x', y')) \mid |x - x'| + |y - y'| = 1\}$.

La figure 4.3 suivante représente la grille à 3 lignes et 5 colonnes.

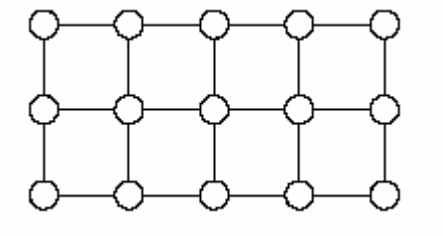


Figure 4.3 – La grille 3×5

Théorème 4.5 [36] La largeur arborescente de la grille $p \times q$ est :

$$tw(M_{p,q}) = \min\{p , q\}.$$

Théorème 4.6 [36] La longueur arborescente de la grille $p \times q$ est :

$$tl(M_{p,q}) = \begin{cases} p-1 & si \quad p = q \quad et \quad p \quad est \quad premier \\ min\{p,q\} & sinon. \end{cases}$$

On a le tableau suivant [36] qui représente la largeur et la longueur arborescente de quelques classes de graphes connues.

Graphes	**Largeur arborescente tw(G)**	**Longueur arborescente tl(G)**
Triangulés	ω(G) – 1	1
Cycles de longueur k	2	$\lceil k/3 \rceil$
Planaires extérieures de cordalité k	2	$\lceil k/3 \rceil$
La grille p×q ($M_{p\times q}$)	min{p,q}	p – 1 si p = q et p impair min{p,q} sinon

Ainsi, les graphes triangulés sont de largeur ω(G)-1 (où w(G) représente le cardinal de la plus grande clique de G), et de longueur 1 (car les graphes triangulés sont des arbres de cliques, et le diamètre d'une clique est toujours égale à 1).

4.4 Approximation de la longueur arborescente d'un graphe quelconque

Yon Dourisboure dans [36] a proposé deux algorithmes et une heuristique pour montrer qu'il est possible de calculer pour tout graphe une décomposition arborescente de longueur proche du minimum.

4.4.1 Une 3 – approximation avec l'algorithme Lex-M

Lex-M utilise un ordre lexicographique basé sur un parcours en largeur du graphe en entrée. Pendant le parcours, les sommets sont numérotés par des entiers entre 1 et n. Par la suite, $\alpha(i)$ représente le sommet numéroté i. Tout sommet u possède également une étiquette, notée label (u), qui est un ensemble de valeurs prises parmi $\{1,\ldots,n\}$ et classées par ordre décroissant. On définit un ordre sur les étiquettes de la manière suivante ; Soient deux étiquettes $L_1 = [p_1,...,p_k]$ et $L_2 = [q_1,...,q_t]$, on dit $L_1 < L_2$ s'il existe j tel que $p_i = q_i$ pour tout $i<j$ et $p_j < q_j$, ou alors si $k < l$ et que $p_i = q_i$ pour tout $i = 1,\ldots,k$.

Comme H est une triangulation de G, alors pour tout $i \in \{1,\ldots, n\text{-}1\}$ le voisinage de $\alpha(i)$ dans le graphe $H[\alpha(i),\ldots., \alpha(n)]$ est une clique. α est donc un ordre d'élimination parfait sur H.

Soit T un arbre de clique de H. T est une décomposition arborescente de G et sa est ; longueur(T) = $\max_{u,v \in F} \{\text{dist}_G(u, v)\}$.

T étant calculable en temps linéaire, on considère que Lex-M renvoie T.

Algorithme Lex-M

Entrée : Un graphe G = (V,E)

Sortie : Une triangulation de G ; H = (V, E ∪ F)

début

Mettre toutes les étiquettes ainsi que l'ensemble F à ∅ ;

pour *i allant de n à* 1 **faire**

Sélection :

Choisir un sommet u non numéroté d'étiquette maximum ;

Affecter à u le numéro i : $\alpha(i) = u$;

Ajout :

pour tout sommet non numéroté v tel qu'il existe une chaîne

$w_1,w_2,\ldots w_{p+1}$ entre $u = w_1$ et $v = w_{p+1}$ avec w_j non numéroté et

label(w_j) < label(v) pour tout $j \in \{2,\ldots, p\}$ **faire**

Ajouter i dans l'étiquette de v ;

Ajouter l'arête uv dans F ;

fin

fin

fin

Borne supérieure

Proposition 4.4 [36] Soient G un graphe de cordalité k et T la décomposition arborescente calculée par l'algorithme LexM, alors longueur(T) ≤ k/2.

Lemme 4.2 [36] *Soient u, v deux sommets tels que uv* $\in F$ *et x le voisin de v de numéro maximum. Il existe* x' *avec* $\alpha^{-1}(u) < \alpha^{-1}(x') \leq \alpha^{-1}(x)$ *tel qu'il existe une chaîne* $x' = x_1, x_2, \ldots, x_{q+1} = v$ *avec* $x_j w_k \notin E$ *pour tout* $j \in \{2,\ldots,q+1\}$ *et tout* $k \in \{2,\ldots,p\}$.

Proposition 4.4 [36] Soient G un graphe et T la décomposition arborescente calculée par l'algorithme LexM, alors longueur(T) ≤ 3 . tl(G) + 1.

Théorème 4.7 [36]

Grâce à l'algorithme Lex-M, il est possible de calculer en temps O($n.m$) une décomposition arborescente T d'un graphe G telle que :

- longueur(T) ≤ k/2 ou k est la cordalité de G ;
- longueur(T) ≤ 3. tl(G) + 1.

De plus, pour tout i> 1, il existe un graphe de longueur arborescente 2i tel que cette décomposition arborescente est de longueur 6i – 1.

4.4.2 Une 3 – approximation en temps linéaire avec l'algorithme BFS - Layering

L'algorithme BFS – Layering qu'on va présenter est une adaptation d'un algorithme de Chepoi, Dragan et Yan dans [26], qui construit en temps linéaire O($nk + m$) un sous graphe couvrant (1, k + 1) – approximant avec 2n – 2 arêtes pour tout graphe k – chordal.

Définition 4.3 Pour tout sommet *u* d'un graphe G et tout sous ensemble de sommets *X*, on définit out (*u*, *X*), l'ensemble des sommets de *X* pour lesquels, il existe une chaîne menant à *u* dont tous les sommets intermédiaires ne sont pas dans *X*.

Soit *s* un sommet d'un graphe G, on note C^i le cercle de rayon *i* centré en *s* : $C^i = \{u \in V / dist_G(s, u) = i\}$ et B^i la boule de rayon *i* centrée en *s* : $B^i = \bigcup_{j=0}^{i} C^i$.

Un partitionnement en couches de G est un partitionnement de chaque C^i en $C_1^i,....,C_{p_i}^i$ tel que $u, v \in C_j^i$ si et seulement si out (*u*, B^{i-1}) = out (*v*, B^{i-1}).

Soit H le graphe dont l'ensemble des sommets est un partitionnement en couches d'un graphe G. Dans H, deux sommets C_j^i et $C_{j'}^{i'}$ sont adjacents si et seulement si, il existe $u \in C_j^i$ et $v \in C_{j'}^{i'}$ tels que *u* et *v* sont adjacents dans G.

Lemme 4.3 [27] H est calculable en temps linéaire : O(*n* + *m*) et H est un arbre. H est appelé, un arbre de partitionnement en couches.

Exemple (a) (b) (c)

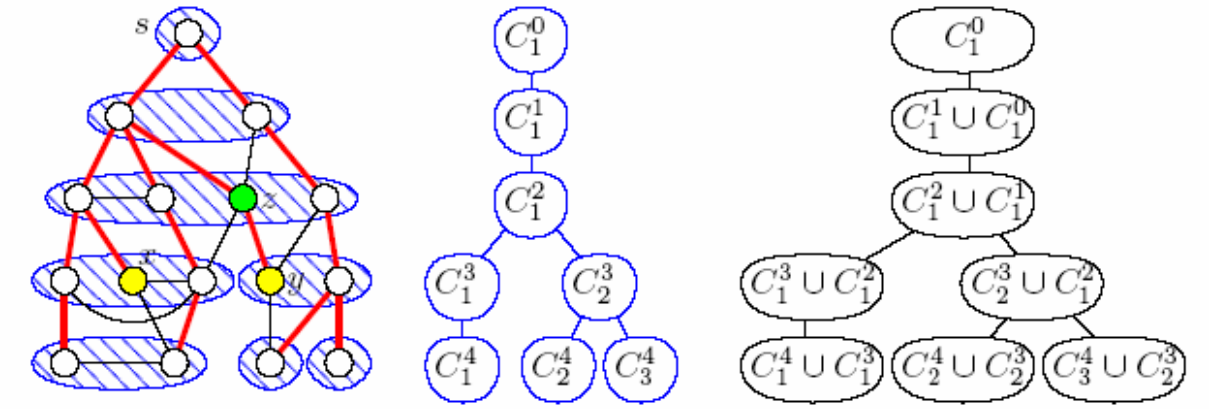


Figure 4.4 – un graphe G , un arbre de partitionnement en couches de G, et *T* une décomposition arborescente de G associée à ce partitionnement.

La figure 4.4 (a) représente un graphe et un partitionnement en 5 couches et 8 partitions de ce graphe. Deux sommets qui sont dans une même partition sont à la même distance du sommet. Par contre, deux sommets, comme les deux sommets *x* et *y*, qui sont à la même distance de *s*, sont dans deux partitions différentes, car toutes les chaînes de l'un à l'autre passent par un sommet plus proche de la source *s*.

La figure 4.4 (b) montre un arbre de partitionnement en couches du graphe G. Cet arbre possède 5 niveaux (il est de profondeur 4) et possède 8 nœuds. Par exemple, les partitions C_1^2 et C_2^3 sont adjacentes dans l'arbre car dans G le sommet *y* et le sommet *z* sont adjacents.

On peut définir maintenant l'algorithme BFS – Layering, qui construit une décomposition arborescente de G à partir d'un arbre de partitionnement en couches (voir figure 4.4 (c)).

Algorithme BFS – Layering

Entrée : Un graphe G

Sortie : Une décomposition arborescente *T* de G.

début

Construire H, un arbre de partitionnement en couches de G ;

Soit *T* une copie de H ;

Pour tout sommet U de T faire

U ← U∪V où V est le parent de U dans H ;

fin

fin

Lemme 4.4 [36] *T* est une décomposition arborescente de G.

Théorème 4.8 [36] L'algorithme BFS – Layering calcule en temps O(*n* + *m*) une décomposition arborescente *T* d'un graphe G telle que :

- longueur(*T*) ≤ *k*/2 + 3 ou *k* est la cordalité de G ;
- longueur(*T*) ≤ 3. tl(G) + 1.

De plus, pour tout *i*> 1, il existe un graphe de longueur arborescente 2*i* tel que cette décomposition arborescente est de longueur 6*i* + 1.

4.4.3 Une 2 – approximation avec l'heuristique Ball – tree ?

L'heuristique Ball – tree est basée sur l'observation que si un graphe G admet une décomposition arborescente *T* de longueur *k* , alors tout sac de *T* est inclus dans la boule de rayon *k* centrée sur n'importe quel sommet du sac.

Définition 4.4 L'ensemble des sommets de G couverts par T , est noté covered(T) : covered(T) = $\bigcup_{X \in V(T)} X$. Celui des sommets qui sont couverts et qui ont au moins un voisin non couvert, est noté border (T) : border (T) = $\{u \in$ covered (T) $\mid$ N(u)$\not/$ covered (T) $\neq \varnothing\}$.

Dans un sous ensemble S de sommets de G, deux sommets u et v sont dits en conflit si $v \in$ out (covered (T) $\cup$ S, u) et si $dist_G(u, v) > k$. Lorsque c'est le cas, il faut retirer un des sommets. Pour décider lequel, on enlève, on utilise la fonction de coût suivante :

$$cost(S,u) = \begin{cases} \infty \;\; si \;\; u \in border(T) \\ 0 \;\; s'il \;\; n'existe \;\; pas \;\; de \;\; sommets \;\; en \;\; conflit \;\; dans \;\; S / \{u\} \\ max_{x,y \in S} \{dist_G(x,y) / x,y \;\; sont \; en \; conflit \;\; dans \;\; S / \{u\}\} \;\; sinon \end{cases}$$

L'arbre T est construit en profondeur, c'est-à-dire qu'un sommet c peut être choisi si et seulement si il est dans border (T) et pour tout v $\in$ border (T), depth (B(c)) $\geq$ depth (B(v)). L'ensemble des sommets choisis est noté C.

Heuristique Ball- tree

Entrée : Un graphe G = (V, E) et un entier k

Sortie : soit ECHEC, soit T : une décomposition arborescente de G.

Début

Soit u est un sommet quelconque de G ;

$T \leftarrow \{u, \varnothing\}$ C $\leftarrow \{u\}$;

Tant que covered (T) $\neq$ V(G) **faire**

Init :

Choisir c$\in$C, si c n'existe pas alors renvoyer (ECHEC) ;

S $\leftarrow$ Composante connexe de c dans G[V(G)/ (covered (T) / {c})] ;

S $\leftarrow$ S $\cap$ B^k(c), où B^k(c) est la boule de rayon k centrée en c ;

S $\leftarrow$ S $\cup$ (out(c, covered (T)) $\cap$ border (T)) ;

Reduce :

Tant que il existe des sommets en conflit dans S **faire**

Soient u, v deux sommets en conflit dans S tels que $dist_G(u, v)$ = $max_{x, y \in S}\{dist_G(x, y) / x, y$ sont en conflit$\}$;

si cost (S, u) < cost (S, v) **alors**

retirer u de S ;

sinon retirer v de S ;

fin

Update : **Si** S $\not\subset$ covered (T) **alors**

$T \leftarrow$ (V(T) $\cup$ {S}, E(T) $\cup$ {$\langle$B(s), S$\rangle$}) ;

C $\leftarrow$ {$u \in$ border (T) / $\forall$ $v \in$ border (T), depth (B(u)) $\geq$ depth (B(v))} ;

Sinon retirer c de C ;

fin

Renvoyer (T) ;

fin

Proposition 4.5 [36] L'heuristique Ball – tree se termine en temps polynomial. De plus lorsqu'il se termine avec succès, T est une décomposition arborescente de G de longueur au plus 2k.

4.5 Minimisation simultanée de la longueur et de la largeur d'une décomposition arborescente

On a vu, qu'il était possible de construire pour les graphes planaires extérieurs, une décomposition arborescente qui est minimum à la fois pour la longueur et pour la largeur. Il en est de même pour la grille à p lignes et q colonnes lorsque p $\neq$ q ou lorsque p est pair.

Alors, est – il toujours possible de construire une décomposition arborescente qui minimise à la fois la longueur et la largeur ? De manière générale, une décomposition arborescente T d'un graphe est α - optimale si longueur (T) $\leq \alpha$. tl(G) et largeur (T) $\leq \alpha$. tw(G).

Alors la question devient : Existe – t – il une constante α telle que tout graphe G admet une décomposition arborescente α - optimale ?

Malheureusement, il n'existe pas encore de réponse à cette question, mais des exemples existent pour certains graphes, où on a trouvé un compromis entre la longueur et la largeur d'un décomposition arborescente.

4.5.1 Cas des pastèques

Définition 4.5 La pastèque de dimension *p, q* est le graphe composé de *p* chaînes de longueur *q* dont les deux extrémités sont confondus. Autrement dit c'est le graphe biparti complet $K_{2, p}$ dont la moitié des arêtes ont été subdivisées en chaînes de longueur $q - 1$.

La figure ci-dessous montre, comment à partir d'un graphe biparti complet $K_{2, 4}$ et en remplaçant chaque arête incidente à *v* par une chaîne de longueur 5, on obtient la pastèque de dimension 4, 6.

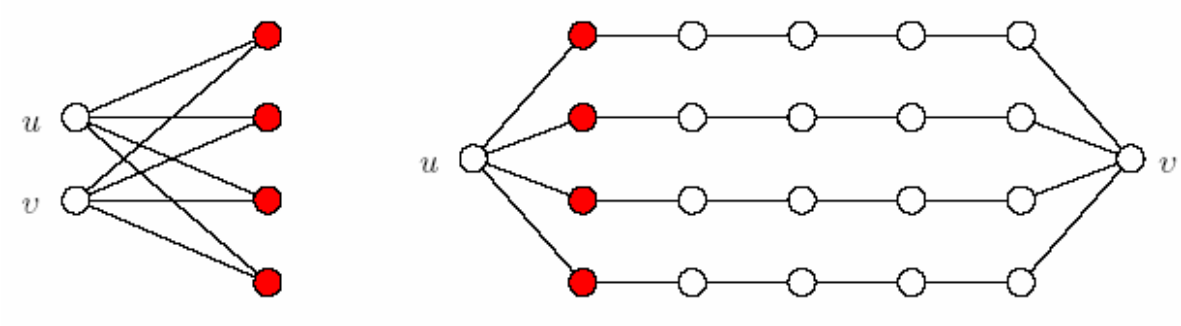


Figure 4.5 – Le graphe biparti complet $K_{2, 4}$ et la pastèque de dimension 4, 6.

Proposition 4.5 [36] Pour tout *p, q* ≥ 2 et G une pastèque de dimension *p, q* est de largeur arborescente 2 et de longueur arborescente $\lceil \frac{2q}{3} \rceil$.

Donc, les pastèques étant de diamètre *q*, toute décomposition arborescente est de longueur au plus $q = \frac{3}{2}$ tl(G). En particulier toute décomposition arborescente qui est de largeur minimum est 1.5 – optimale.

CHAPITRE 5

Calcul de la longueur arborescente

Nous sommes capables de déterminer en temps linéaire si un graphe est de longueur arborescente 1 [33]. En effet, il suffit de tester s'il est triangulé. Cependant, peut – on savoir quels sont les graphes qui sont de longueur arborescente bornée par 2 ?

Dans ce chapitre, nous allons montrer que les graphes faiblement triangulés répondent à la question.

De plus, nous allons faire le lien entre le calcul de la longueur arborescente et les triangulations minimales des graphes, et enfin nous allons montrer que les graphes *k*-cordaux sont de longueur arborescente bornée par $\lceil k/3 \rceil$.

5.1 Longueur arborescente des graphes faiblement triangulés

Remarque 5.1 Dans le chapitre 2, on a vu que :

Toute décomposition arborescente dans un graphe faiblement triangulé (retournée par l'algorithme de calcul de la largeur arborescente) a ses noeuds (ou sacs) étiquetés par les cliques maximales potentielles Ω qui sont de la forme $(N(x) \cap N(y)) \cup \{x\}$ ou $(N(x) \cap N(y) \cup \{y\}$ avec *x* et *y* formant une deux paire dans le graphe .

Proposition 5.1 *Tout graphe faiblement triangulé est de longueur arborescente ≤ 2.*

Avant de démontrer la proposition, on va redéfinir la longueur arborescente pour les graphes qui admettent une famille complète de cliques maximales potentielles $\Im$;

Définition 5.1 Le diamètre d'un sac Ω est la distance dans le graphe entre les deux sommets les plus éloignés du sac.

$$\text{diam}_G\ \Omega = \text{Max}_{u,v \in \Omega} \{\text{dist}_G(u, v)\}$$

La longueur d'une décomposition arborescente *T* est le plus grand diamètre de ses sacs ;

$$\text{longueur}(T) = \text{Max}_{\Omega \in V(T)} \{\text{diam}_G(\Omega)\}$$

La longueur arborescente d'un graphe G notée tl(G) est la plus petite des longueurs de toutes les décompositions arborescentes possibles de G

$$\text{tl}(G) = \text{Min}_T \{\text{longueur}(T)\}$$

Preuve

Soit G un graphe faiblement triangulé.

1er cas

Si G est triangulé, alors tl(G) = 1

2ème cas

G un graphe non triangulé avec $|V| > 3$ (sinon G est triangulé)

Supposons par l'absurde que tl(G)> 2

Donc tl(G) = Min_T {longueur(T)}> 2 $\Rightarrow \forall\ T$, longueur $T > 2$

Ie : $\text{Max}_{\Omega \in V(T)}$ {$\text{diam}_G(\Omega)$}> 2 $\Rightarrow \exists\ \Omega \in V(T)$ / $\text{diam}_G(\Omega) > 2$

Ie : $\exists \Omega_0 \in V(T)$ / $\text{Max}_{\Omega 0 \in V(T)}$ {$\text{diam}_G(\Omega)$}> 2 $\Rightarrow \exists\ u\ ,\ v \in \Omega_0$ / $\text{dist}_G(u,v) > 2$

On suppose sans perte de généralité que $\text{dist}_G(u,v) = 3$ (avec $u,v \in \Omega_0$)

Ω est de la forme $(N(x) \cap N(y) \cup \{x\})$ ou $(N(x) \cap N(y) \cup \{y\})$

Le même raisonnement s'applique aux deux ensembles, raisonnons sur $(N(x) \cap N(y) \cup \{x\}$

- si $x = u$ et $v \in N(x) \cap N(y)$

  On aura $\text{dist}_G(u,v) = 1$ (absurde)

- si $x = v$ et $u \in N(x) \cap N(y)$

  On aura $\text{dist}_G(u,v) = 1$ (absurde)

- si $x \neq u$ et $x \neq v$ alors $u\ ,\ v \in N(x) \cap N(y)$

  Donc u et v sont tous deux adjacents à x et on aura $\text{dist}_G(u,v) = 2$ (absurde)

On en conclut que $\text{dist}_G(u,v) \leq 2 \Rightarrow \text{diam}_G(\Omega) \leq 2 \Rightarrow$ longueur $T \leq 2$.

$\Rightarrow$ tl(G) ≤ 2 . Et comme G est non triangulé, alors tl(G) = 2

Dans tous les cas, si G est faiblement triangulé alors tl(G) ≤ 2 .

Corollaire 5.1

On peut déterminer pour un graphe s'il est de longueur arborescente ≤ 2 en temps polynomial en $O(n^2m)$

Preuve

En effet , il suffit de montrer s'il est faiblement triangulé et l'algorithme de reconnaissance des graphes faiblement triangulés est en $O(n^2m)$.

Corollaire 5.2

Les graphes bipartis - cordaux, les graphes HHD – free et les graphes à distances héréditaires sont de longueur arborescente ≤ 2.

Preuve

Ce sont des sous classes des graphes faiblement triangulés.

Corollaire 5.3

Pour tout graphe faiblement triangulé, on peut construire en temps polynomial une décomposition arborescente qui est à la fois de largeur et de longueur minimum.

Preuve

Par l'algorithme d'élimination et d'extraction pour le calcul de la largeur arborescente des graphes faiblement triangulés (voir chapitre 2).

Dans le tableau suivant, nous donnons la largeur et la longueur arborescente des graphes cités, la largeur étant calculable en temps polynomial sauf pour les graphes à distances héréditaires, où la largeur est calculable en temps linéaire, et la longueur arborescente de ces graphes est bornée par 2.

graphes	***Largeur arborescente tw(G)***	***La borne de la longueur arborescente tl(G)***
Faiblement triangulés	polynomial	2
Bipartis - cordaux	polynomial	2
HHD - free	polynomial	2
Distances héréditaires	linéaire	2

Routage compact dans les graphes faiblement triangulés

Ainsi, de ce qui précède, on obtient le résultat suivant sur le routage compact dans la classe des graphes faiblement triangulés ;

Proposition 5.2 *Les graphes faiblement triangulés admet un schéma de routage de déviation 4, sans boucles, et indépendants de la numérotation des ports. Dans ces schémas, les adresses et les mémoires locales sont de taille O(2 log^3 n) bits par sommet.*

Preuve

En effet les graphes faiblement triangulés sont de longueur arborescente bornée par 2 , donc en appliquant le théorème 4.3 pour les graphes faiblement triangulés , on obtient la proposition 5.2.

5.2 Le calcul de la longueur arborescente en passant par les triangulations minimales

Dans cette partie, nous allons étudier l'algorithme LB-Triang développé par A. Berry et Al. [15], ensuite on l'adaptera pour le calcul de la longueur arborescente de tout graphe.

5.2.1 LB – Triangulation

Notations

G = (V,E) connexe , $X \subseteq V$, $C_G(X)$: ensemble des composantes connexes de G(V \ X) .

On note N[x] , l'ensemble $N(x) \cup \{x\}$.

$S \subset V$ est dit séparateur si $|C_G(S)| \geq 2$, un *a,b* - séparateur si a et b sont dans différentes composantes connexes de $C_G(S)$, *a,b* - séparateur minimal si pour {*a,b*} il existe des composantes connexes pleines C_1 et $C_2 \in C_G(S)$ tels que $N(C_1) = N(C_2) = S$

$\mathcal{J}_G(S)$: ensemble des séparateurs minimaux de G

Une sous étoiles S de x est un sous ensemble de N(x) tel que S = N(C) pour une composante connexe de G(V \ N[x]).

On sait que les sous étoiles sont précisément les séparateurs minimaux de x inclus dans le voisinage de x.

a) Graphes triangulés et triangulations

Caractérisation 5.1 [15] Un graphe G est dit triangulé si et seulement si tout séparateur minimal de G est une clique.

Définition 5.2 Un sommet *x* est dit LB – simplicial si et seulement si tout séparateur minimale inclus dans son voisinage est une clique.

Caractérisation 5.2 [15] Un graphe G est triangulé si et seulement si tout sommet est LB – simplicial.

b) Triangulations minimales : Processus algorithmique de base

Nous utilisons la caractérisation 5.2 pour calculer les triangulations minimales en forçant tout sommet à être LB – simplicial par une addition locale d'arêtes. On prouve que la triangulation obtenue est minimale en montrant que tel processus choisit et sature un ensemble de séparateurs minimaux deux à deux parallèles du graphe en entrée.

Algorithme LB – Triang [15]

Entrée : Un graphe G = (V, E)
Sortie : Une triangulation minimale de G

Pour tout $x \in V$ faire
    Rendre *x* LB – simplicial
Fin

A la fin de l'exécution, $\alpha = (x_1, x_2, \ldots, x_n)$ est l'ordre dans lequel les sommets ont été traités, et G_α^{LB} va être le graphe triangulé obtenu.

On note que l'algorithme traite les sommets dans un ordre arbitraire, ainsi tout ordre peut être choisi.

Définition 5.3 La déficience d'un sommet x dans un graphe G, notée $Def_G(x)$ est un ensemble d'arêtes qui a été ajouté à G pour rendre x simplicial.
On définit la LB – déficience d'un sommet $x \in V$ comme étant l'ensemble d'arêtes ajouté à G pour le rendre LB – simplicial.
Clairement, pour tout graphe, LB $Def_G(x) \subseteq Def_G(x)$, $\forall x \in G$.

On note G_i : le graphe à l'étape i, x_i est le sommet traité durant l'étape i, F_i : l'ensemble de complétion des arêtes ajoutées à l'étape i pour rendre x_i LB – simplicial dans G_i.
$\mathcal{J}_i$: ensemble des séparateurs minimaux inclus dans $N_{G_i}(x_i)$.Ainsi $F_i = LBDef_{G_i}(x_i)$ et G_{i+1} est le graphe obtenu à partir de G en ajoutant un ensemble d'arêtes F_i (ou de façon équivalentes en saturant les séparateurs de $\mathcal{J}_i$).
Rendre un sommet x_i LB – simplicial (par la définition 5.2) requière un ensemble $\mathcal{J}_i$ de séparateurs minimaux inclus dans $N_{G_i}(x_i)$

Propriété 5.1 [15] Pour un sommet $x \in G$, l'ensemble des séparateurs minimaux de G inclus dans N(x) est exactement $\{N(C) \ / \ C \in C(N[x])\}$.

Par conséquent, pour calculer l'ensemble d'arêtes F_i , ajoutées à G_i pour rendre x_i LB – simplicial dans le graphe résultant G_{i+1}, on suit les trois étapes suivantes :
- Calculer $N_{G_i}[x_i]$.
- Calculer les composantes connexes C dans $C_{G_i}(N_{G_i}[x_i])$.
- Calculer les voisins $N_{G_i}(C)$ pour tout C.

L'une des propriétés intéressante de l'algorithme LB – Triang est que :
Quand x_i est LB – simplicial dans G_{i+1}, il va rester LB – simplicial tout le reste du processus et aussi être LB – simplicial dans G_α^{LB} .

Exemple d'application de l'algorithme LB – triang

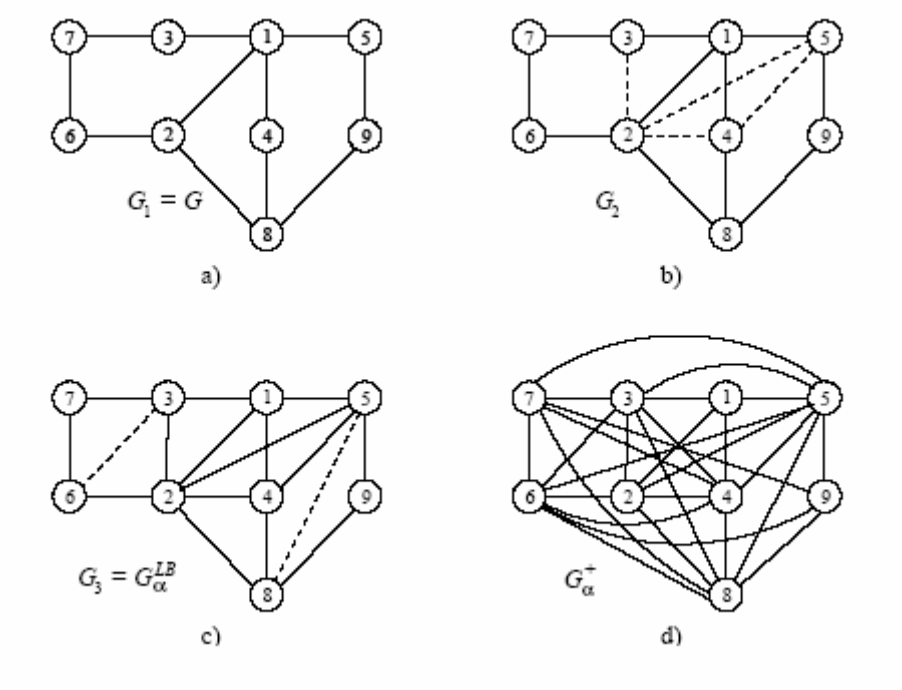


Figure 5.1 – Exemple d'application de l'algorithme LB – Triang.

Théorème 5.1 [15] L'algorithme LB – Triang calcule les triangulations minimales du graphe en entrée.

c) Propriétés importantes de LB – Triang

Théorème 5.2 [15] L'algorithme LB – Triang calcule la même complétion indépendamment du fait (de supprimer ou non) tout sommet LB – simplicial à la fin de chaque étape de l'algorithme.

Corollaire 5.4 [15] Le schéma d'élimination parfait de LB – triang calcule une triangulation minimale du graphe en entrée.

Théorème 5.3 [15] Etant donné un graphe G et une triangulation minimale H de G, il existe un ordre α des sommets de G telle que $G_\alpha^{LB} = H$.

5.2.2 Calcul de la longueur arborescente

On se propose maintenant de répondre à la question suivante :
Peut – on calculer en temps polynomial la longueur arborescente d'un graphe quelconque ?
Et plus précisément peut – on montrer en temps polynomial si la longueur arborescente d'un graphe G donné est égale à δ ?

Définition 5.4 Une décomposition arborescente D_T de G = (V, E) est une paire (X , *T*) où T = (I , F) est un arbre avec un ensemble de nœuds I et un ensemble d'arêtes F.
Et $X = \{X_i, i \in I\}$ est une famille de sous ensembles de V tels que :

1) $\bigcup_{i \in I} X_i = V$

2) $\forall vw \in E \ , \exists X_i \ , i \in I \quad avec \quad v \in X_i \quad et \quad w \in X_i$

3) $\forall i, j, k \in I \ , si \ \ j \ est \ \ sur \ \ une \ \ chaine \ de \ i \ à \ k \ \ dans \ T \quad alors$

$$X_i \bigcap X_k \subseteq X_j$$

Proposition 5.2 Pour tout graphe G, il existe une triangulation H^* telle que :

$$tl(H^*) = tl(G).$$

Preuve : Soit G = (V , E) un graphe de longueur arborescente tl(G).
Considérons une décomposition arborescente ($\{X_i, i \in I\}$, *T*) du graphe G.
Par définition les ensembles $T_x = \{ i \in I / x \in X_i \}$ forment des sous arbres de *T*.
On considère maintenant le graphe H = (V , F) ou $xy \in F$ ssi *x* et *y* sont dans une même étiquette X_i.
Clairement la famille $\{ T_x , x \in V \}$ des sous arbres de *T* est un modèle d'intersection de H et par (le théorème de Gavril (1974)) on obtient que H est un graphe triangulé.
Par la 2ème condition de la définition d'une décomposition arborescente, toute arête de G est aussi une arête de H.

Par conséquence H est une triangulation de G, H = (V, E $\cup$ E$^{/}$) avec $LongueurT_H = \max_{u,v \in E^{/}} \{dist_G(u,v)\}$ et tl(G) = tl(H*) / H* la triangulation correspondante à la décomposition arborescente de longueur minimum.

Calculer la longueur arborescente (tree – length) d'un graphe G est équivalent à trouver ladite triangulation minimale H* qui minimise le diametre des sacs B. C'est-à-dire qui minimise la distance dans le graphe entre les deux sommets les plus éloignés du sac.
Où bien globalement, trouver la triangulation minimale qui minimise dans le graphe la distance entre les extrémités de toute arête dans E$^{/}$ / H = (V , E $\cup$ E$^{/}$).
C'est-à-dire est – ce qu'il existe un ordre α / G_α^{LB} est une triangulation minimale telle que ; tl(G_α^{LB}) = tl(G).

Remarque 5.2

On sait que pour un cycle de longueur *k* , tl(G) = $\lceil k/3 \rceil$.
L'algorithme LB – Triang avec un certain ordre α_0 peut donner des triangulations minimales de longueur $\lceil k/3 \rceil$ comme suit :

Algorithme

Entrée : Un graphe cycle de longueur *k*

Sortie : Une triangulation minimale de longueur $\lceil k/3 \rceil$

Choisir $x \in V$ / $\forall$ *u, v* $\in$ LBDef(*x*), $dist_G(u , v) \leq \lceil k/3 \rceil$

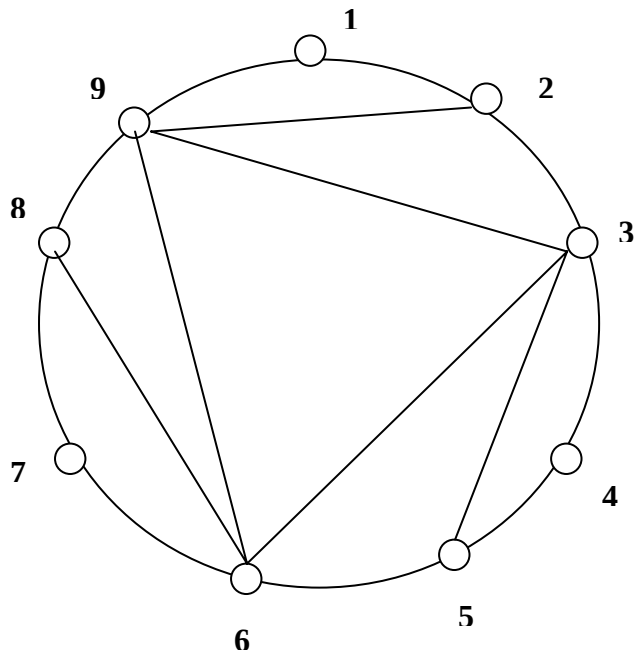
Rendre *x* LB – simplicial

Fin

Preuve

Soit G ; le cycle de longueur *k*
On sait que tl(G) = $\lceil k/3 \rceil$
D'après la proposition 5.1 , $\exists$H* / tl(H*) = tl(G) = $\lceil k/3 \rceil$.
Donc, il existe un ordre α_0 / tl(G) = tl($G_{\alpha_0}^{LB}$) et cet ordre est tel que ;
$\forall$ *u, v* $\in$ LBDef(*x*), $dist_G(u , v) \leq \lceil k/3 \rceil$

Exemple



Pour le cycle l'algorithme est optimal.

Figure 5.2

Triangulation minimale de longueur minimum
$\lceil k/3 \rceil = \lceil 9/3 \rceil = 9$

Algorithme de calcul de tl(G) pour les graphes k-cordaux

Entrée : Un graphe G = (V, E) k- cordal

Sortie : Triangulation minimale de longueur minimum.

Choisir $x \in V$ / $\forall u, v \in LBDef(x)$, $dist_G(u , v) \leq \lceil k/3 \rceil$.

Rendre x LB – simplicial.

Fin

Justification de l'algorithme

Premièrement, on montre qu'on obtient une triangulation.

Notons, « $\forall u, v \in LBDef(x)$, $dist_G(u , v) \leq \lceil k/3 \rceil$ » comme étant la propriété (P).

Lemme 5.1 [15] Soit G un graphe, et x un sommet de G. Les séparateurs minimaux inclus dans N(x) sont deux à deux parallèles dans G.

Lemme 5.2 [15] Soit G un graphe, $G^{/}$ le graphe de G en saturant un ensemble de séparateurs minimaux deux à deux parallèles de G, et soit x un sommet LB-simplicial de G. Alors $N_G(x) = N_{G^{/}}(x)$.

Lemme 5.3 [15] Soit G un graphe, $G^{/}$ le graphe de G en saturant un ensemble de séparateurs minimaux deux à deux parallèles de G, et soit x un sommet LB-simplicial de G . Alors x est LB-simplicial dans $G^{/}$.

Lemme 5.4 Soit G un graphe, $G^{/}$ le graphe de G en saturant un ensemble de séparateurs minimaux deux à deux parallèles de G, et soit x un sommet LB-simplicial de G qui vérifie la propriété (P). Alors x est LB-simplicial dans $G^{/}$ et vérifie la propriété (P).

Preuve : x LB-simplicial dans G $\Rightarrow$ x est LB-simplicial dans $G^{/}$ (lemme 5.4).
x vérifie la propriété (P) dans G $\Rightarrow$ x vérifie la propriété dans $G^{/}$.
Car $\forall\ u, v \in LBDef_G(x)$, on a $dist_G(u , v) \leq \lceil k/3 \rceil$.
Dans $G^{/}$, on a $\forall\ u, v \in LBDef_{G/}(x)$, $dist_{G/}(u , v) = 1$ (tous les séparateurs minimaux inclus dans $N_G(x)$ deviennent des cliques dans $G^{/}$).
Mais, il reste que $\forall\ u, v \in LBDef_{G/}(x)$, on a $dist_G (u , v) \leq \lceil k/3 \rceil$.

Maintenant, on va prouver ;
Lemme 5.5 Durant une exécution de l'algorithme, tout sommet qui est LB-simplicial et vérifie la propriété (P), reste LB-simplicial et vérifie la propriété (P) dans toutes les autres étapes.
Preuve : Pour i = 1 à n, par le lemme 5.1, G_{i+1} est obtenu à partir de G_i en saturant un ensemble de séparateurs minimaux deux à deux parallèles de G_i, par le lemme 5.4 tout sommet LB-simplicial qui vérifie la propriété (P) dans G_i reste LB-simplicial et vérifie la propriété (P) dans G_{i+1}.

Lemme 5.6 Le graphe G_α^{LB} résultant de l'algorithme est une triangulation de G.

Preuve : Par le lemme 5.5, à la fin de l'exécution, tout sommet de G_α^{LB} est LB-simplicial. Par la caractérisation 5.3 , G_α^{LB} est triangulé

Lemme 5.7 [15] Pour tout i de 1 à n+1, l'ensemble $\cup_{1\leq j\leq i} \mathcal{J}_i$ des séparateurs minimaux saturés jusqu'à l'étape i est un ensemble de séparateurs minimaux deux à deux parallèles dans G.

Lemme 5.8 [15] Soit G un graphe et G/ le graphe obtenu de G en saturant un ensemble de séparateurs minimaux deux à deux parallèles de G. Si G/ est triangulé, alors G/ est une triangulation minimale de G.

Théorème 5.4 L'algorithme calcule une triangulation minimale du graphe en entrée, de plus tout sommet vérifie la propriété (P).

Preuve : Du lemme 5.6, on obtient un graphe qui est triangulé, et par le lemme 5.7, G_α^{LB} est obtenu à partir de G en saturant un ensemble de séparateurs minimaux deux à deux parallèles de G. Par le lemme 5.8, G_α^{LB} est une triangulation minimale de G. De plus, tout sommet x de G vérifie la propriété (P).

Lemme 5.9 [15] Soit G = (V,E) un graphe, et a,b,y ∈ V. L'arête ab appartient à LBDef(y) si et seulement si il existe un cycle sans cordes a, y, b, x_1, ….,x_k, a, avec k≥1 dans G.

Théorème 5.5

Tout graphe k-cordal est de longueur arborescente égale à ⌈k/3⌉.

Preuve :

Soit H = G_α^{LB} la triangulation minimale obtenue par l'algorithme et notons par *T* un arbre de cliques de H. On a que les cliques de *T* sont de la forme $x_i \cup \mathcal{J}_i$ où $\mathcal{J}_i$; l'ensemble des séparateurs minimaux inclus dans le voisinage de x_i..

Supposons que tl(H) > ⌈k/3⌉ alors on a :

longueur(*T*) > ⌈k/3⌉

Donc, Longueur (*T*) = Max $_{B\in V(T)}$ {$diam_G(B)$}> ⌈k/3⌉

Et donc, il existe B tel que ; Max $_{u,v\in B}$ {$dist_G$ (u , v)}> ⌈k/3⌉.

Ainsi, ∃ u_0, v_0 ∈ LBDef(x_i) , $dist_G(u_0, v_0)$ > ⌈k/3⌉. (absurde).

Supposons maintenant que tl(H) < ⌈k/3⌉.

Alors, on a ∀*x*∈V , ∀ *u v* ∈ LBDef(*x*), $dist_G(u , v)$ < ⌈k/3⌉

$u\ v \in$ LBDef(x) $\Leftrightarrow$ $u, v \in N_G(x)$, $u \neq v$, $uv \notin$ E, et il existe une chaîne sans cycle de u à v, les sommets intermédiaires appartiennent à V\ N[x] .(lemme 5.9)

Le cycle est donc ; ($u, x , v, y_1,\ldots,y_l, u$), le cycle est de longueur $dist_G(u, x) + dist_G(x, v) + dist_G(v, u) < \lceil k/3 \rceil + \lceil k/3 \rceil + \lceil k/3 \rceil < k$.

Donc, pour tout x, x appartient à un cycle sans cordes de longueur < k. Donc, G est un graphe de cordalité < k . (absurde).

On en déduit le résultat suivant sur le routage compact dans la classe des graphes k-cordaux

Proposition 5.4 *Tout graphe k-cordal admet un schéma de routage de déviation 2.* $\lceil k/3 \rceil$ *, sans boucles, et indépendant de la numérotation des ports. Dans ce schéma, les adresses et les mémoires locales sont de taille O(* $\lceil k/3 \rceil . \log^3 n$*) bits par sommet.*

Preuve

En effet les graphes k - cordaux sont de longueur arborescente égale à $\lceil k/3 \rceil$, donc en appliquant le théorème 4.3 pour les graphes k - cordaux , on obtient la proposition 5.4.

<u>Algorithme de calcul de tl(G) pour les graphes quelconques</u>.

Entrée : Un graphe G = (V, E) et un entier $\delta \geq 1$.

Sortie : Triangulation minimale de longueur δ ou une réponse tl(G) > δ

Choisir $x \in V$ / $\forall\ u, v \in$ LBDef(x), $dist_G(u , v) \leq \delta$.

Si x existe ; rendre x LB – simplicial.

Sinon, répondre tl(G) > δ.

<u>Fin</u>

Justification de l'algorithme

On sait que l'algorithme LB – Triang donne une triangulation minimale de n'importe quel graphe et ce, en utilisant n'importe quel ordre.

Donc si l'algorithme termine, il donnera une triangulation minimale de G , sinon l'algorithme répondra que tl(G) > δ

Il reste à montrer que tl(G_α^{LB}) = δ .

C'est évident car à chaque étape i , on aura $\forall\ u, v \in$ LBDef(x_i), $dist_G(u , v) \leq \delta$

Conclusion et perspectives

Dans cette thèse, en faisant le lien entre le calcul de la longueur arborescente des graphes et les triangulations minimales, nous avons montré que :

1. La longueur arborescente des graphes faiblement triangulés est bornée par 2.
2. La longueur arborescente des graphes k - cordaux est égale à $\lceil k/3 \rceil$

Mais, il reste certains points ouverts qui peuvent faire l'objet d'études futures, nous citons par exemples :

- Existe – t – il en théorie des graphes, des problèmes difficiles en général, mais qui seraient faciles à résoudre dans les graphes de longueur arborescente bornée ?

- Quelle est la longueur arborescente des graphes planaires ?

Références

[1] H.Ait Haddadene and M.A.Boutiche, *Graphs of treelength at most two, and tree decomposition of minimum length for k-chordal graphs* , In preparation.

[2] S.Arnborg, D.G.Corneil, and A.Proskurowski, *Complexity of finding embedding in k-trees*, SIAM Journal on Algebraic and Discrete Methods, 8 (1987), pp.277-284.

[3] S.Arnborg, J.Lagergren, and D.Seese, *Easy Problems for tree-decomposable graphs* Journal of Algorithms, 12 (1991), pp.308-340.

[4] B.Awerbuch, A.Bar-Noy, N.Linial, and D.Peleg, *Improved routing strategies with succint tables*, Journal of Algorithms ,11 (1990), pp. 307-341.

[5] C.Berge, *Graphes*, Gauthier-Villars, 3ème ed., 1983.

[6] A. Berry, *Désarticulation d'un graphe*, Thèse (PhD Dissertation), LIRMM, Montpellier, décembre 1998 .

[7] A. Berry, *Weaving Through a Crowd of Minimal Separators*, HDR (Habilitation à Diriger les Recherches), LIMOS , Clermont-Ferrand Dec 2003.

[8] A.Berry and J.P.Bordat, *Triangulated and Weakly Triangulated Graphs: Simpliciality i Vertices and Edges***.** Communication, 6th International Conference on Graph Theory (ICGT 2000).

[9] A. Berry, M. C. Golumbic and M. Lipsteyn, *Two tricks to Triangulate Chordal Probe Graphs in Polynomial Time***.** Proceedings SODA 2004.

[10] A. Berry, G. Simonet and P. Heggernes, *The Minimum Degree Heuristic and the Minimal Triangulation Process***.** Proceedings WG 2003 - 29th Workshop on Graph Theoretic Concepts in Computer Science, June 2003, Elspeet, the Netherlands. Springer Verlag, Lecture Notes in Computer Science 2880, pages 58 - 70.

[11] A. Berry, Jean Blair and Pinar Heggernes, *Maximum Cardinality Search for Computing Minimal Triangulations***.** Proceedings WG 2002 - 28th Workshop on Graph Theoretical Concepts in Computer Science, Cesky Krumlov, Czech Rebublic, June 2002. Springer Verlag, Lecture Notes in Computer Science 2573, pages 1-12.

[12] A. Berry and J-P.Bordat, *Decomposition by clique minimal separators***.** Dagstuhl Seminar No. 01251 Report No. 312 : 17.06.2001-22.06.2001 on Graph Decompositions and Algorithmic Applications .

[13] A. Berry, J-P.Bordat and Pinar Heggernes, *Recognizing Weakly Triangulated Graphs by Edge Separability***.** Proceedings of the Seventh Scandinavian Workshop on Algorithm Theory (SWAT'2000).

[14] A. Berry and J.-P. Bordat, *Local LexBFS Properties in an Arbitrary Graph***.** Proceedings of Journées Informatiques Messines (JIM 2000).

[15] A. Berry, J-P. Bordat, P. Heggernes, G. Simonet, and Y. Villang, *A wide-range algorithm for minimal triangulation from an arbitrary ordering***.** Journal of Algorithms, *in press, to appear*.

[16] A. Berry, J. Blair, P. Heggernes and B. Peyton, *Maximum Cardinality Search for Computing Minimal Triangulations of Graphs*, Algorithmica 39-4 (2004), pages 287 - 298.

[17] A. Berry, J-P.Bordat and Pinar Heggernes, *Recognizing Weakly Triangulated Graphs by Edge Separability***.** Nordic Journal Computing, vol.7, no.3, pp. 164-177, 2000.

[18] J.R.S.Blair and B.W.Peyton, *On finding minimum diameter clique trees*, Nordic Journal of Computing, 1 (1994), pp. 173-201.

[19] H.L.Bodlaender, *A linear time algorithm for finding tree decomposition of small treewidth*, SIAM Journal on Computing, 25 (1996), pp. 1305-1317.

[20] H.L.Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoretical Computer Science, 209 (1998), pp. 1-45.

[21] H.L.Bodlaender, abd T.Hagerup, *Tree decomposition of small diameter*, in 23rd International Symposium on Mathematical Foundation of Computer Sciences (MFCS), vol. 1450 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 702-712.

[22] H.L.Bodlaender, D.M.Thilikos, *Treewidth and small separators for graphs with small chordality*, Discrete Applied Mathematics, 79 (1997), pp. 45-61.

[23] N.Bonichon, C.Gavoille, and N.Hanusse, *Canonical decomposition of outerplanar maps and application to enumeration, coding and generation*, Research Report RR-1295-03, LaBRI, University of Bordeaux, 351, cours de la Libération, 33405 Talence Cedex, France, Mar.2003.

[24] V.Bouchitté, D.Kratch. H.Muller, and I.Todinca, *On treewidth approximation*, Discrete Applied Mathematics, 136 (2004), pp. 183-196.

[25] V.Bouchitté and I.Todinca, *Listing all Potential Maximal Cliques of a graph,* Theoret. Comput. Sci. 276 1-2 (2002) pp. 17-32.

[26] V.Bouchitté and I.Todinca, *Treewidth and minimum fill-in : grouping the minimal separators,* SIAM J . Comput. 31 (1) (2001) pp. 212-232.

[27] V.D.Chepoi, and D.Dragan, *A note on distance approximating trees in graphs*, European J. Combinatorics, 21 (2000), pp. 761-768.

[28] V.D.Chepoi, D.Dragan and C.Yan, *Additive spanner for k-chordal graphs*, in Algorithms and Complexity : 5th Italian Conference (CIAC 2003), vol. 2653 of L ecture Notes in Computer Science, Springer-Verlag, Heidelberg, May 2003, pp. 96-107.

[29] B.Courcelle, *The monadic second order-logic of graphs* III : *Tree-width, forbidden minors and complexity issues*, Informatique Théorique, 26 (1992), pp. 257-286.

[30] B.Courcelle and M.Mosbah, *Monadic second order evaluations on tree-decomposable graphs*, Theoretical Computer Science, 109 (1993), pp. 49-82.

[31] R.Diestel, *Graph Theory* (second edition), vol 173 of Graduate Texts in Mathematics, Springer, Feb. 2000.

[32] G.Dirac, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp.71-76.

[33] Y.Dourisboure, *Routage compact et longueur arborescente*, Thèse de doctorat, Université Bordeaux I, (2003).

[34] Y.Dourisboure, *An additive stretched routing scheme for chordal graphs*, in 28-th International Workshop on Graph Theoretic Concepts in Computer Science (WG '02), vol. 2573 of Lecture Notes in Computer Science, Springer-Verlag, June 2002, pp. 150-163.

[35] Y.Dourisboure and C.Gavoille, *Improved compact routing scheme for for chordal graphs*, in 6-th International Symposium on DIStributed Computing (DISC 2002), vol. 2508 of Lecture Notes in Computer Science, Springer-Verlag, Oct. 2002, pp. 252-264.

[36] Y.Dourisboure and C.Gavoille, *Tree-decomposition of graphs with small diameter bags*, in European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB 2003), ITI, Sept. 2003.

[37] P.Fraignaud and C.Gavoille, *Universal routing scheme*, Journal of Distributed Computing, 10 (1997), pp. 65-78.

[38] P.Fraignaud and C.Gavoille, *Routing in trees*, in 28-th International Colloquium on Automata, Languages and Programming (ICALP), F.Orejas, and J.V.Leeuwen, eds, vol.2076 of Lecture Notes in Computer Science, Springer, July 2001, pp. 757-772.

[39] P.Fraignaud and C.Gavoille, *A space lower bound for routing in trees*, in 19-th Annual Symposium on Theoretical Aspects of Computer Science (STACS), vol. 2285 of Lecture Notes in Computer Science, Springer, Mar 2002, pp. 65-76.

[40] C.Gavoille and S.Pérennés, *Memory requirement for routing in distributed networks*, in 15-th Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM PRESS, May 1996, pp. 125-133.

[41] F.Gavril, *The intersection graphs of a path in a tree are exactly the chordal graphs*, Journal Comb Theory, 16 (1974), pp. 47-56.

[42] M.C.Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, New York, (1980).

[43] R.Hayward, *Weakly triangulated graphs*, J.Comb.Theory, Ser B, 39 (1985), pp. 200-208.

[44] R.Hayward , C.Hoang and F.Maffray, *Optimizing weakly triangulated graphs*, Graphs Combin, 5 (1989), pp. 339-349.

[45] M.Habib and R.H.Möhring, *Treewidth of cocomparability graphs and a new order-theoretic parameter*, ORDER, 11 (1994), pp. 47-60.

[46] C.W.Ho and R.C.T.Lee, *Counting clique trees and computing perfect elimination schemes in parallel*, Inform. Process. Letters, 31 (1989), pp. 61-86.

[47] R.Khatib and N.Santoro, *Labelling and implicit routing in networks*, The Computer Journal, 28 (1985), pp. 5-8.

[48] T.Kloks, D.Kratch, and H.Müller, *Approximating the bandwidth for asteroidal triple-free graphs*, in 3-rd Annual European Symposium on Algorithms (ESA '95), vol. 979 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 434-447.

[49] M.Lundquist, Zero patterns, *chordal graphs and matrix completion*, 1990. PhD Thesis, Clemson University.

[50] L.Narayanan and N.Nishimura, *Interval routing on k-trees*, Journal of Algorithms, 26 (1998), pp. 325-369.

[51] A.Parra, *Structural and algorithmic aspects of chordal graphs embeddings*, 1996. PhD Thesis, Technische Universität, Berlin.

[52] A.Parra and D.Scheffler, *Characterizations and algorithmic applications of chordal graphs embeddings*, Discrete Applied Mathematics, 79 (1997), pp. 171-188.

[53] D.Peleg, *Proximity-preserving labeling schemes and their applications,* in 25th International Workshop, Graph – Theoretic concepts in Computer Science (WG), vol. 1665 of Lecture notes in Computer Science, Springer, June 1999, pp. 30-41.

[54] N.Robertson and P.D.Seymour, *Graph minors*. II. *Algorithmic aspects of tree-width*, Journal of Algorithms, 7 (1986), pp. 309-322.

[55] D.Rose, R.E.T.arjan and G.S.Luecker, *Algorithmic aspects of vertex elimination on graphs*, SIAM Journal on Computing, 5 (1976), pp. 266-283.

[56] R.E.Tarjan and M.Yannakakis, *Simple linear time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM Journal on Computing, 13 (1984). pp. 647-655.

[57] M.Thorup and U.Zwick, *Approximate distance oracles*, in 33rd Annual ACM Symposium on Theory of Computing (STOC), Hersonissos, Crete, Greece, July 2001, pp.183-192.

[58] M.Thorup and U.Zwick, *Compact routing scheme*, in 13rd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), Hersonissos, Crete, Greece, July 2001, ACM PRESS, pp.1-10.

[59] I. Todinca, *Aspects algorithmiques des triangulations minimales des graphes*, Thèse de Doctorat, ENS. Lyon, 1999.