N° d'ordre : 09/2013-D/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE (ALGER)

FACULTE DE MATHÉMATIQUES

THÈSE

Présentée pour l'obtention du grade de DOCTEUR

en : **MATHÉMATIQUES**

Spécialité : **RECHERCHE OPÉRATIONNELLE**

Par : **BOUTICHE Mohamed Amine**

Thème :

TECHNIQUES DE DECOMPOSITIONS DE GRAHES APPLIQUEES AU PROBLEME DE ROUTAGE

Soutenue publiquement, le 12 Décembre 2013, devant le jury composé de :

M. AIDER Méziane, Professeur à l'USTHB	Président
Mme. LE THI Hoai An, Professeur, à l'UL Lorraine, Metz (France)	Directrice de Thèse
M. AIT HADDADENE Hacène, Professeur, à l'USTHB	Co-Directeur de Thèse
M. CHELLALI Mustapha, Professeur, à l'USD Blida	Examinateur
M. SADI Bachir, Maître de Conférences Classe A, à l'UMM Tizi Ouzou	Examinateur
M. SEMRI Ahmed, Maître de Conférences Classe A, à l'USTHB	Examinateur

Mis en page avec la classe thesul.

Remerciements

Je tiens dans un premier temps à remercier mon directeur de thèse, Monsieur **Hacène AIT HADDADÈNE**, Professeur à l'USTHB, pour m'avoir confié ce travail de recherche, ainsi que pour son aide et ses précieux conseils au cours de ces années.

Je tiens aussi à exprimer ma reconnaissance et toute ma gratitude envers ma directrice de thèse, Madame **Hoai An LE THI**, Professeure des universités de classe exceptionnelle à l'Université de Lorraine, qui m'a soutenue tout au long de ces années, pour sa disponibilité, ses idées et conseils, ainsi que pour son aide précieuse dans la réalisation de cette Thèse.

Je voudrais aussi présenter mes remerciements très sincères au président du jury Monsieur **Méziane AIDER**, Professeur à l'USTHB, qui m'a fait l'honneur de présider mon jury de Thèse.

J'adresse aussi un grand remerciement à Monsieur **Mustapha CHELLALI**, Professeur à l'université de Blida, Monsieur **Bachir SADI**, Maitre de conférence à l'Université de Tizi Ouzou et Monsieur **Ahmed SEMRI**, Maitre de conférence à l'USTHB d'avoir accepté de lire mon travail et de faire partie du jury de cette Thèse.

Je remercie également, le MINISTÈRE ALGÉRIEN DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE qui a financé cette thèse en m'accordant un stage de longue durée au sein du Laboratoire d'Informatique Théorique et Appliquée de Metz. Je remercie toutes les personnes que j'ai rencontré à Metz : Mes amis Belaid, Abdallah, Samir, Abdelkader, Yacine, Mohamed, Lyes, Quinh, Minh, Tuy, Cuong, Mamadou, sans oublier Vincent avec qui j'ai partagé un bureau durant un an et demi.

Je remercie aussi sincèrement mes amis et collègues de l'USTHB Mohammed, Fayçal, Khaled, Mourad et Lyes qui m'ont soutenu tout au long de ses années.

J'adresse enfin mes remerciements à toutes celles et à tous ceux que j'ai côtoyé, durant ces dernières années et qui m'ont aidé de prés ou de loin à l'élaboration de cette Thèse.

Je dédie cette thèse
À ma famille.

Table des matières

Introduction générale

La théorie des graphes est un outil très puissant pour la modélisation de problèmes réels dans beaucoup de domaines. En recherche opérationnelle, les graphes sont étudiés parce qu'ils constituent de bonnes structures de données et parce qu'ils permettent de visualiser un problème en vue de le résoudre. Les décompositions donnent souvent une vision plus précise de la structure d'un graphe. Leur objectif consiste à représenter le graphe par un ensemble de graphes, qui sont souvent des sous-graphes induits du premier, ou en sont très proches. Cette approche vise à "morceler" le graphe en de plus petites parties qui permettent de mieux mettre en exergue leurs propriétés. La plupart des décompositions (dont celle présentée dans cette thèse) sont arborescentes. C'est à dire que les relations entre les graphes issus de la décomposition forment un arbre. Cette propriété améliore grandement la lisibilité du résultat et joue pour beaucoup dans l'intérêt de ces décompositions.

Les décompositions arborescentes (en anglais : tree decomposition) des graphes est un concept fondamental en théorie des graphes et a été introduit par Robertson et Seymour [116] pour la résolution de problèmes très variés. La largeur arborescente (en anglais : tree - width) est un paramètre de graphes largement étudié dans la littérature [3, 19, 98, 116]. Beaucoup de problèmes intraîtables peuvent être résolus en temps polynomial pour les classes de graphes de largeur arborescente bornée [3, 64]. La longueur arborescente (en anglais : tree - length) est un paramètre de graphes récemment introduit par Dourisboure et Gavoille [52]. Cette notion est motivée par le fait que les graphes de longueur arborescente bornée admettent des sous graphes couvrants qui approximent les distances, des schémas de routage et des étiquetages de distances avec un facteur d'éloignement additif (i.e. ; déviation constante par rapport aux plus courts chemins) [50, 49, 104].

La classe des graphes triangulés est l'une des classes les plus étudiée des graphes. L'une des raisons est que cette classe admet une décomposition arborescente naturelle appelée arbre de clique (en anglais : clique tree), qui peut être construite en temps linéaire [65, 50, 68]. Le calcul de la longueur arborescente et de la largeur arborescente pour cette classe est connu (au plus $\omega(G)$ pour la largeur, où $\omega(G)$ est la taille de la plus grande clique, et la longueur arborescente est égal à un). La classe des graphes faiblement triangulés est aussi une classe bien étudiée et a de nombreuses applications connues [35, 77, 123]. Cette classe de graphes a été introduite par Hayward [75] et est une sous classe des graphes parfaits qui inclut les graphes triangulés et leurs complémentaires.

La problématique du routage dans un réseau (de communication ou de télécommunication) consiste à "router" l'information (ou les messages) entre ces composants (ou nœuds) qui peuvent être des stations de travail, machines parallèles, antennes, etc. Les nœuds d'un réseau ne sont pas tous forcément reliés entre eux, car ça engendrerait un coût exorbitant, d'où la nécessité de dôter chaque nœud d'une "certaine connaissance" du réseau, pour qu'il soit capable de décider localement vers où faire transiter le message pour qu'il puisse arriver à destination. Ainsi, la

conception d'architectures capables de prendre en charge cette problématique s'est avérée être un challenge que doit relever la communauté s'intéressant aux problèmes des réseaux. La problématique de conception d'architecture (ou de topologie) de réseau fait apparaître les notions de famille de graphes et de schémas spécifiques à chaque famille, qui relèvent des communautés de l'algorithmique des graphes et de la recherche opérationnelle.

La construction de schémas de routage pour les familles de graphes admettant une décomposition arborescente dont les "sacs" sont de diamètre bornés proposée par Dourisboure [50], a été le point de départ de nos travaux. En effet, la première partie de notre travail a consisté à étudier une nouvelle notion introduite dans la thèse de Dourisboure [50], "la longueur arborescente des graphes" venue compléter celle largement étudiée, la largeur arborescente. Notre contribution a été le calcul de la longueur arborescente pour des classes de graphes plus larges que celles proposées jusque là, dont les principaux résultats ont étés publiés dans [22, 21], et une version papier soumise à une revue internationale [28].

Dans la deuxième partie de notre travail, nous avons considéré que la topologie du réseau est dynamique (i.e ; des modifications sur les liens ou sur les nœuds du réseau surviennent à travers le temps, donc nous avons proposé de maintenir cette topologie à chaque modification. Cette problématique appartient au domaine de l'algorithmique dynamique, qui est opposée à celle de l'algorithmique statique.

En effet, un algorithme statique de graphe doit répondre à une question donnée par un calcul. Par contre, un algorithme dynamique de graphe considère qu'un calcul demandé est déjà effectué (par un algorithme statique), et s'intéresse à mettre à jour (ou actualiser) le résultat à chaque fois que le graphe subit une modification. L'idéal dans ce cas serait d'actualiser le nouveau résultat sans refaire tous le calcul, c'est la solution à moindre coût.

Nous avons proposé dans cette partie un algorithme dynamique qui maintient le réseau représenté par sa décomposition arborescente dans les cas de suppression d'un nœud ou d'une arête et dans les cas de l'ajout d'un nœud ou d'une arête. De plus, nous recalculons les paramètres largeur et longueur arborescente à chaque changement.

Le premier algorithme dynamique qui maintient la représentation arbre de clique des graphes triangulés a été introduit par L. Ibarra [87]. Cet algorithme supporte les opérations de suppression et ajout d'arêtes. Plus tard, dans [29, 26], nous avons généralisé ce résultat, d'abord pour la classe des graphes faiblement triangulés, et ensuite pour les graphes quelconques en présentant un algorithme qui maintien quelques propriétés sur les graphes quelconques.

Cette thèse est organisée de la manière suivante :

Dans le premier chapitre, nous introduisons les différentes notions utilisées en théorie des graphes de manière courante et dont nous nous servons dans ce document. Nous y présenterons également les notions liées aux triangulations minimales des graphes et de décomposition arborescente, concepts fondamentaux et au centre de nos travaux.

Les concepts liés au domaine d'application qui a motivé cette thèse en l'occurence la problématique du routage compact seront présentés dans le deuxième chapitre. Nous donnerons aussi, les principaux résultats obtenus dans la littérature dans ce domaine.

Dans le troisième chapitre, sera présenté les principales techniques de calcul des décompositions arborescentes. Nous présenterons les méthodes exactes, les méthodes heuristiques et les algorithmes minimaux. Ces techniques sont presque toutes liées au calcul des triangulations minimales des graphes. De plus, nous détaillerons de manière non exhaustive, les principaux résultats

liés aux paramètres largeur arborescente et longueur arborescente. On notera qu'on présentera ici notre première contribution déjà étudiée en Magister et approfondie quelques années plus tard. Ce travail concerne le calcul de la longueur arborescente pour les graphes faiblement triangulés et k-cordaux. Ce travail a aussi été valorisé dans [21, 22] et récemment soumis [28].

Le quatrième et dernier chapitre contient des contributions importantes de cette thèse, qui ont fait l'objet de participations à des conférences internationales [23, 24, 25, 27, 30], et d'articles parues dans des journaux [26, 29]. Nous commencerons d'abord par introduire le lecteur avec les notions de l'algorithmique dynamique, concept d'une importance capitale dans cette thèse. On s'intéressera aux algorithmes dynamiques et à leurs implémentations, en comparant les différentes structures de données y afférentes. Ensuite nous donnerons les principaux résultats obtenus au cours de nos travaux. La première contribution est un algorithme dynamique pour les graphes faiblement triangulés et qui supporte les opérations de suppression et d'ajout d'arêtes et de sommets. De plus, nous mettons à jour les paramètres largeur et longueur arborescente quand des changements dans la topologie surviennent. La deuxième contribution est une généralisation pour les graphes quelconques.

Table des figures

Chapitre 1

Préliminaires

Sommaire

Dans ce premier chapitre, nous allons donner les notions élémentaires requises pour une bonne lecture de ce document.

1.1 Graphes et sous-graphes

Définition 1 *Un graphe (ou graphe non orienté) G est défini par un couple $(V(G), E(G))$, où $V(G)$ est un ensemble fini non vide et $E(G)$ une relation binaire*[1] *commutative sur $V(G)$.*

L'ensemble $V(G)$ est appelé l'ensemble des sommets de G, et $E(G)$ l'ensemble des arêtes de G.

Nous utiliserons parfois la notation fonctionnelle $V(G)$ pour désigner l'ensemble des sommets d'un graphe G, et $E(G)$ l'ensemble de ses arêtes. Sauf précision nous désignerons par n et m les cardinaux respectifs de $V(G)$ et $E(G)$. Pour un graphe G donné, nous définissons le vocabulaire suivant :

- Si (u, v) est une arête de G, on dit que (u, v) est incidente aux sommets u et v.
- Deux sommets u et v sont adjacents dans G si l'arête (u, v) appartient à $E(G)$.

1. Une relation binaire entre deux ensembles E et F (ou simplement relation entre E et F) est caractérisée par un sous-ensemble du produit cartésien $E \times F$, soit une collection de couples dont la première composante est dans E et la seconde dans F

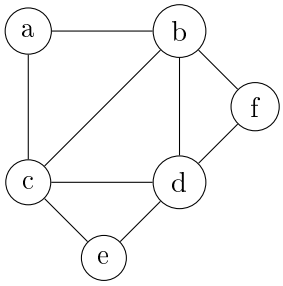


FIGURE 1.1 – Un graphe non orienté de 6 sommets et 9 arêtes

- Pour un sommet u, l'ensemble des voisins de u dans G, noté $N_G(u)$, est l'ensemble des sommets qui lui sont adjacents.
- Le degré d'un sommet u de G, noté $d_G(u)$, est le nombre de sommets qui lui sont adjacents, c'est à dire $|N_G(u)|$.
- Nous notons respectivement $\delta(G)$ et $\Delta(G)$ le plus petit et le plus grand degré parmi les sommets de G.

En l'absence d'ambiguïté, le G disparaît des notations : V, E, $N(u)$, δ, Δ.

Nous considérons dans toute la suite que $G = (V, E)$ est un graphe non orienté, simple et sans boucles.

Définition 2 *Un sous-graphe d'un graphe G est le graphe obtenu en supprimant certains sommets et toutes les ars incidentes aux sommets supprimés.*

*Soit $G = (V, E)$ un graphe, un **sous-graphe** de G est un graphe $G' = (V', E')$ tel que :*

- $V' \subseteq V$.
- $E' \subseteq E \cap \{(u, v) | u, v \in V'\}$.

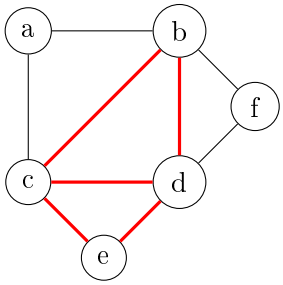


FIGURE 1.2 – Un exemple de sous-graphe induit

Définition 3 *Un graphe partiel d'un graphe G est le graphe obtenu en supprimant certaines arêtes.*

Soit $G = (V, E)$ un graphe, un graphe partiel de G est un graphe $G' = (V, E')$ avec $E' \subseteq E$.

Définition 4 *Une **chaîne** P est un graphe dans lequel :*

- $V(P) = \{x_0, x_1, ..., x_k\}$ *avec* $k \geq 0$.
- $E(P) = \{(x_i, x_{i+1}), i \in \{0, ..., k-1\}\}$.

Une chaîne d'un graphe G est donc un sous-graphe de G. Pour une chaîne P donnée :
- Les extrémités de P sont les sommets x_0 et x_k. Nous dirons que P relie les sommets x_0 et x_k.
- Les sommets x_1 à x_{k-1} sont les maillons internes de P.
- La longueur de P est égale à son nombre d'arêtes $|E(P)|$ (ici égal à k).

Définition 5 *Un graphe non vide $G = (V, E)$ est dit **connexe** si toute paire de sommets est reliée par une chaîne de G. Une composante connexe de G est un sous-graphe connexe de cardinalité maximale. Un graphe connexe possède une seule composante connexe.*
Un graphe est dit non connexe s'il possède au moins deux composantes connexes.

Définition 6 *Un cycle C est un graphe dans lequel :*
- $V(C) = \{x_0, x_1, ..., x_k\}$, *avec* $k \geq 0$,
- $E(C) = E(P) \cup \{(x_k, x_0)\}$ *où* $E(P) = \{(x_i, x_{i+1}), i \in \{0, ..., k-1\}\}$.

Un **cycle** d'un graphe G est donc une chaîne de G ayant comme extrémités le même sommet x_0. La longueur d'un cycle est égale à son nombre d'arêtes $|E(C)|$ (ici égal à $k+1$).

Définition 7 *Une **corde** dans un graphe G est une arête reliant deux sommets non adjacents appartenant au même cycle.*

Définition 8 *La **distance** entre deux sommets u et v de G, notée $dist_G(u, v)$ est la longueur de la plus courte chaîne de u à v.*

Définition 9 *Le **diamètre** d'un graphe connexe G noté $Diam(G)$, est la plus grande distance entre deux de ses sommets, i.e., $Diam(G) = \max\{dist_G(u, v)/u, v \in V(G)\}$.*

Définition 10 *Une **clique** est un ensemble de sommets où toute paire de sommets est reliée par une arête. On dit que c'est un graphe complet.*

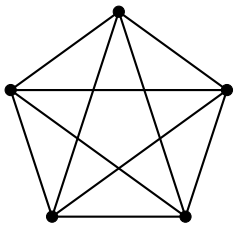


FIGURE 1.3 – Une clique à 5 sommets

Définition 11 *Un **stable** est un ensemble de sommets où aucune paire de sommets n'est reliée par une arête. Un stable est un graphe formé de sommets isolés.*

1.2 Arbres et arbres couvrants

Définition 12 *Un **arbre** est un graphe connexe sans cycles.*

Théorème 13 *[10] Pour un graphe G les propriétés suivantes sont équivalentes :*

- *G est un arbre ;*
- *G est sans cycles et possède n sommets et $m = n - 1$ arêtes ;*
- *G est connexe et possède n sommets et $m = n - 1$ arêtes ;*
- *G est connexe et on le déconnecte en retirant une arête quelconque ;*
- *G est connexe et on crée un unique cycle en ajoutant une arête quelconque ;*
- *Pour toute paire $\{x, y\}$ de sommets de G, il existe une chaîne unique d'extrémités x et y.*

Remarque 14 *On peut orienter d'une manière unique un arbre en arborescence en choisissant une racine parmi n'importe lequel de ses sommets. Cette arborescence sera appelée arbre enraciné.*

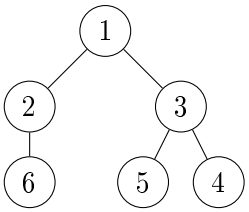


FIGURE 1.4 – Un arbre enraciné sur le nœud 1

- 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, 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'enfants 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 3 a deux enfants : les nœuds 4 et 5, ils sont donc frères. Les nœuds $4, 5, 6$ sont des feuilles et l'arbre est de profondeur 2 car les chaînes entre les feuilles $4, 5$ ou 6 et la racine 1, sont de longueur 2. Le plus petit ancêtre commun entre les nœuds 4 et 6 est le nœud 1.
- 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 traîtés avant n'importe quel nœud de profondeur $i + 1$. Dans l'arbre de la figure 1.4, un parcours en profondeur peut traîter successivement les nœuds $1, 2, 6, 3, 5, 4$ alors qu'un parcours en largeur pourra les traiter dans l'ordre $1, 2, 3, 6, 5, 4$.

Définition 15 *Un sous graphe G' d'un graphe G est un **arbre couvrant** de G 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, $dist_G(u, v) = dist_{G'}(u, v)$.*

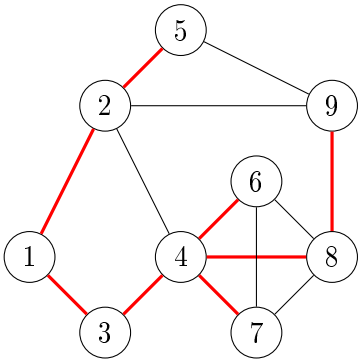


FIGURE 1.5 – Un arbre couvrant de plus courtes chaînes enraciné sur le nœud 3

1.3 Graphes triangulés, faiblement triangulés et graphes k - cordaux

Les graphes triangulés ("chordal graphs" en anglais), sont les graphes sans cycles induits de longueur supérieure ou égale à 4. Autrement dit, tout cycle de longueur au moins 4 possède une corde. Les graphes triangulés sont les graphes d'intersection [2] de sous-arbres d'un arbre. On a le théorème suivant.

Théorème 16 *[129, 65, 33]*
Soit $G = (V, E)$ un graphe. Les trois propositions suivantes sont équivalentes :

1. *G est un graphe triangulé*
2. *G est le graphe d'intersection d'une famille de sous-arbres d'un arbre.*
3. *Il existe un arbre T dont l'ensemble des sommets K est l'ensemble des cliques maximales de G et tel que pour tout $v \in V$, le sous-graphe de T induit par les cliques maximales de G contenant v est connexe, c'est à dire un sous-arbre de T.*

Un arbre T vérifiant la condition 3 du théorème précédent est appelé un arbre de cliques de G.

Théorème 17 *[65] Un graphe est triangulé si et seulement si il admet un arbre de cliques.*

Lemme 1.3.1 *[65]*
Les arbres de cliques sont les modèles d'intersection minimaux des graphes triangulés.

Comment représenter l'ensemble de ces modèles minimaux ? Une représentation communément utilisée est le graphe des cliques de G. Les sommets de ce graphes sont les cliques maximales de G et les arêtes relient deux cliques dont l'intersection est non vide, elles sont valuées par le cardinal de cette intersection.

2. Formellement, un graphe d'intersection est un graphe non orienté formé à partir d'une famille d'ensembles S_i, $i = 0, 1, 2, ...$ en créant un sommet v_i pour chaque ensemble S_i, et en connectant deux sommets v_i et v_j par une arête à chaque fois que les deux ensembles correspondants ont une intersection non vide, qui est, $E(G) = \{(v_i, v_j) | S_i \cap S_j \neq \emptyset\}$

Théorème 18 *[5]*
Les arbres de cliques d'un graphe triangulé sont exactement les arbres couvrants de poids minimum de son graphe de cliques.

Définition 19 *Un sommet x est dit **simplicial** dans un graphe G si et seulement si son voisinage $N(x)$ est une clique.*

Définition 20 *Un ordre total $(x_1, x_2, ..., x_n)$ sur les sommets du graphe $G = (V, E)$ est appelé **ordre d'élimination parfait** (perfect elimination ordering) si pour tout $i \in \{1, ..., n\}$, le sommet x_i est simplicial dans $G(x_i, ..., x_n)$.*

Théorème 21 *[118, 125] Un graphe est triangulé si et seulement s'il admet un ordre d'élimination parfait.*

Définition 22 *Un trou dans graphe est un cycle sans cordes de longueur supérieure à quatre. Un graphe est dit sans trou (hole-free) quand il n'a aucun trou.*

Définition 23 *[75]*
*Un graphe G est dit **faiblement triangulé** si G et $\overline{G}$ sont sans trou.*

Hayward qui a défini cette classe de graphes, a proposé pour elle un algorithme de reconnaissance (voir[76]). Il existe plusieurs algorithmes en $O(m^2)$ de reconnaissance (voir [78, 7]). Ces graphes présentent des propriétés similaires à celles des graphes triangulés [7, 6] qu'ils généralisent.
Par ailleurs, nous avons besoin de définir la classe des graphes k-cordaux.

Définition 24 *Un graphe est dit k-cordal si et seulement si il ne possède aucun cycle sans cordes induit de longueur $k + 1$ ou plus.*

En particulier, les graphes triangulés sont 3-cordaux et les graphes faiblement triangulés sont 4-cordaux.

1.4 Triangulations et triangulations minimales

La triangulation est un processus algorithmique qui consiste à ajouter des arêtes à un graphe afin de le rendre triangulé. Quand cet ajout d'arêtes est minimal, on dira que la triangulation est minimale. La recherche d'une triangulation minimum d'un graphe est malheureusement un problème NP-Complet [125]. Le concept de triangulation minimale qui consiste à ajouter un nombre strictement nécessaire pour la triangulation des graphes a été défini par Ohtsuki, Cheung et Fujisawa [111], et largement étudié par Rose, Tarjan et Luecker [118].

Définition 25 *Soit $G = (V, E)$ un graphe non triangulé. On appelle triangulation minimale du graphe G, tout graphe $H = (V, E + F)$, tel que pour toute partie propre F' de F, le graphe $H' = (V, E + F')$ n'est pas triangulé.*

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

Théorème 26 *[118]*
Pour tout graphe G, il est possible de construire en temps $O(nm)$ une triangulation minimale de G.

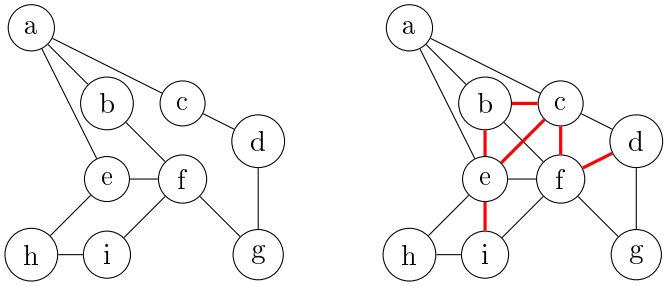


FIGURE 1.6 – Un graphe non triangulé et sa triangulation minimale H

1.5 Séparateurs minimaux

La notion de séparateur minimal a été introduite par Dirac en 1961 [48] pour caractériser la famille des graphes triangulés. Cette notion a été généralisée à des graphes quelconques et largement étudiée durant ces deux dernières décennies par les travaux de Kloks, Kratsch et Spinrad [97, 95], de Parra et Scheffler [112], de Todinca [127], ainsi que par Berry et Bordat [11, 7].

Cette notion est si puissante qu'elle a fourni des résultats théoriques et algorithmiques majeurs pour la décomposition des graphes (voir chapitre 3).

Définition 27 *Soit $G = (V, E)$ un graphe connexe.*

- *On appelle **séparateur** dans G tout sous ensemble de sommets propre S de V dont le retrait déconnecte le graphe (autrement dit $G(V - S)$ possède au moins deux composantes connexes).*
- *Pour un couple de sommets $a, b \in V$, un séparateur S est dit $ab-$**séparateur** si dans $G(V - S)$ les sommets a et b se trouvent dans deux composantes connexes distinctes. De plus, un $ab-$séparateur sera dit minimal si aucun sous ensemble propre de S n'est un $ab-$séparateur.*
- *Un séparateur S est dit minimal s'il existe deux sommets a et b tels que S soit un $ab-$séparateur minimal.*

La définition d'un séparateur peut se généraliser aux graphes non connexes en considérant comme séparateur d'un graphe non connexe tout séparateur d'une composante connexe du graphe.

La définition d'un séparateur se traduit par les chaînes comme suit :

Caractérisation 28 *[112] Soient $G = (V, E)$ un graphe et $x, y \in V - S$.*

- *S est un $xy-$séparateur si et seulement si toute chaîne reliant x à y passe par au moins un sommet de S.*
- *De plus, S est minimal si et seulement si tout sommet de S appartient à une chaîne reliant x et y et ne passant par aucun autre sommet de S.*

On peut réduire un séparateur jusqu'à obtenir un séparateur minimal par le corollaire suiant

Corollaire 1.5.1 *[112] Soient $G = (V, E)$ un graphe et x, y deux sommets de G et $S \subset V$. Si S est un $xy-$séparateur alors il existe un $xy-$séparateur minimal $R \subseteq S$.*

Corollaire 1.5.2 *[112] Soit $G = (V, E)$ un graphe connexe. Si x, y sont deux sommets non adjacents de G, alors G possède un $xy-$séparateur minimal.*

Il est important de noter qu'un séparateur qui est minimal pour la séparation d'une paire de sommets données, n'est pas forcément minimal pour la séparation d'une autre paire de sommets, ni même séparateur.

Définition 29 *Soit S un séparateur d'un graphe $G = (V, E)$ et soit C une composante connexe de $G(V - S)$. C sera dite* ***composante connexe pleine*** *pour S si $N(C) = S$.*

Caractérisation 30 *[11] Soit $G = (V, E)$ un graphe. Un séparateur $S \subset V$ de G est un séparateur minimal si et seulement si $G(V - S)$ admet au moins deux composantes connexes pleines.*

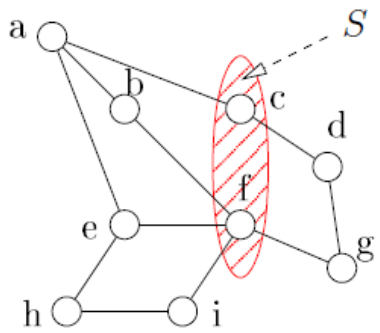


FIGURE 1.7 – Un exemple de séparateur minimal $S = \{c, f\}$

Définition 31 *[97]*
Soit $G = (V, E)$ un graphe et soient S et R deux séparateurs minimaux de G. On dit que R croise S quand il y a dans $G(V - S)$ deux composantes connexes différentes C_1 et C_2 telles que $R \cap C_1 \neq \emptyset$ et $R \cap C_2 \neq \emptyset$.

Caractérisation 32 *[111] Un séparateur minimal d'un graphe G est complet si et seulement si il ne croise aucun autre séparateur minimal de G.*

Propriété 33 *[112] Soit $G = (V, E)$ un graphe. Un séparateur minimal de G, et G_s le graphe obtenu en saturant[3] S dans G. R est un séparateur minimal de G_s si et seulement si R est un séparateur minimal de G qui ne croise pas R dans G.*

Notons que si S croise R alors R croise S, i.e, cette relation est symétrique voir [112].

3. On dit qu'on sature un ensemble de sommets X d'un graphe G si on rajoute les arêtes nécessaires pour rendre $G(X)$ clique

1.6 Triangulations minimales et séparateurs minimaux

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

Proposition 1.6.1 *[96]*
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 $H[V-S]$.

Dans sa thèse Parra [111] a montré que :

Lemme 1.6.1 *[111]*
Les relations de parallélisme et de croisement sont symétriques entre séparateurs minimaux.

Par exemple dans la Figure 1.8, 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.

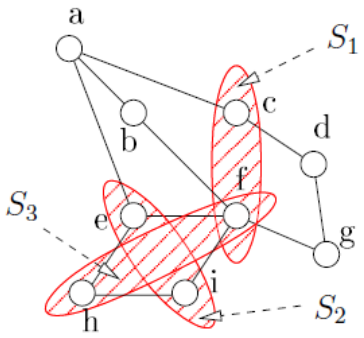


FIGURE 1.8 – Croisement et parralèlismes entre séparateurs minimaux

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

Théorème 34 *[112]*
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.

1.7 Mineurs de graphes

Soit (x, y) une arête d'un graphe $G = (V, E)$. On note par $G\ (x, y)$ le graphe obtenu à partir de G en contractant l'arête (x, y) et en supprimant toutes les boucles et les arêtes parallèles résultantes. On note par $G > H$ si un graphe isomorphe [4] à H peut être obtenu à partir d'un sous graphe de G par contraction d'arêtes. On dit dans ce cas que H est un mineur de G. Donc,

4. Soit $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ deux graphes. Un isomorphisme de graphe de G_1 dans G_2 est une bijection ϕ de E_1 dans E_2 qui vérifie : $(i, j) \in E_1 \iff (\phi(i), \phi(j)) \in E_2$

si H est un mineur de G, alors G admet un sous graphe G' tel que $V(G')$ peut être partitionné en $|H|$ sous-ensembles disjoints et connexes dans G, indicés par les sommets de H et il y a au moins une arête entre deux sous-ensembles associés à deux sommets adjacents de H. Un graphe G est dit sans mineur H, si le graphe H n'est pas un mineur de G. Par exemple, la figure 1.9, représente le graphe de Petersen P_{10}. En contractant les deux arêtes rouges et en ne gardant que les arêtes bleues, et les sommets adjacents aux arêtes bleues, on obtient un $K_{2,4}$. Donc le graphe de Petersen admet $K_{2,4}$ comme mineur.

Théorème 35 *[117]*
Pour tout graphe fixé H et pour tout graphe G d'ordre n, il existe un algorithme, qui en $O(n^3)$ décide si un graphe G contient H comme mineur.

Par ailleurs, nous avons le théorème suivant :

Théorème 36 *[128]*
Un graphe G est planaire si et seulement si G ne contient ni K_5 ni $K_{3,3}$ comme mineur.

Définition 37 *Un graphe sans mineur $K_{2,3}$ et K_4 est dit planaire-extérieur .*

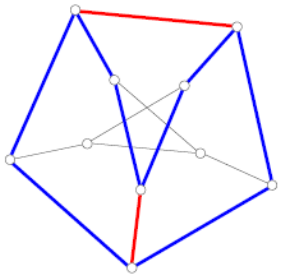


FIGURE 1.9 – Construction d'un mineur $K_{2,4}$ à partir du graphe de Petersen P_{10}

1.8 Complexité algorithmique

Dans cette section, nous allons présenter un nombre de classes de complexité dans lesquelles les problèmes de décision et d'existence se situent. Nous renvoyons le lecteur intéressé par la complexité algorithmique vers les ouvrages de Garey et Johnson [62], et de Paschos [113], et un récent survey d'un point de vue logique et algorithmique de Samuel Buss [34] pour une littérature plus détaillée sur le sujet.

Définition 38 *Un problème est dit dans la **classe** P s'il peut être résolu par un algorithme déterministe en temps polynomial par rapport à la taille de l'instance. On qualifie alors le problème de polynomial, c'est un problème de complexité $O(n^k)$ pour un certain k.*

La **classe** NP des problèmes non-déterministes polynomiaux réunit les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. De façon équivalente, c'est la classe des problèmes pour lesquels il existe un algorithme polynomial déterministe A, et pour chaque instance positive I un certificat $cert(I)$, tels que l'algorithme A sait reconnaître à l'aide de $cert(I)$ que la réponse est oui pour I. Intuitivement, ce sont les problèmes qui peuvent être résolus en énumérant l'ensemble des solutions possibles et en les testant à l'aide d'un algorithme polynomial. Le nombre de solutions peut néanmoins être exponentiellement grand.

On a l'inclusion $P \subseteq NP$, mais on ne sait pas si cette inclusion est stricte, ou si $P = NP$. Une grande part de la théorie de la complexité s'est construite sous l'hypothèse que $P \neq NP$.

Définition 39 *La **classe** NPO est formée des problèmes d'optimisation qui vérifient les propriétés suivantes :*

- *La faisabilité d'une solution est vérifiable en temps polynomial,*
- *La valeur d'une solution réalisable est calculable en temps polynomial,*
- *Une solution réalisable est calculable en temps polynomial.*

Proposition 1.8.1 *[113] La valeur décisionnelle d'un problème d'optimisation dans NPO appartient à NP.*

Définition 40 *La **classe NP - complet** est une sous-classe de NP constituée des problèmes de décision qui sont les plus difficiles de NP, et de difficulté équivalente par rapport à leur résolution polynomiale. Plus formellement, un problème de décision Π est NP - complet si et seulement s'il vérifie ces deux conditions :*

1. $\Pi \in NP$,
2. $\forall \Pi' \in NP$, Π' *se réduit à* Π *par un algorithme polynomial.*

Un problème qui vérifie la condition 2 (sans pour autant la condition 1) est appelé NP-**difficile** (NP-hard dans la littérature anglaise). On peut donc dire qu'un problème est NP-complet si et seulement s'il appartient à NP et est NP-difficile.

Ainsi, pour démontrer qu'un problème Π est NP-complet, il suffit de démontrer les deux propriétés suivantes :

- $\Pi \in NP$,
- Un problème quelconque Π' NP-complet se réduit à Π par un algorithme polynomial.

Pour en terminer avec les classes de complexité, notons que la notion de NP-complétude est propre aux problèmes de décision. Lorsque l'on traite des des problèmes d'optimisation, le terme approprié utilisé dans la littérature francophone, est NP-difficile. Nous avons alors la définition suivante :

Définition 41 *Un problème de NPO est NP-difficile si et seulement si sa variante décisionnelle est un problème NP-complet.*

1.9 Décompositions arborescentes

La notion de décomposition arborescente a été introduite par Robertson et Seymour en 1986 lors de leurs travaux sur les mineurs de graphes [116].

Définition 42 *Une décomposition arborescente d'un graphe $G = (V, E)$ est une paire (S, T) telle que S est l'ensemble $S = \{X_i : i \in I\}$ de sacs et T est un arbre $T = (I, M)$ tel que chaque sac $X_i \in S$ est un sous ensemble de V, de plus les conditions suivantes doivent être vérifiées :*

1. *$\cup_{i \in I} X_i = V$.*
2. *Pour toute arête (u, v) dans E, il existe un sac X_i dans S tel que $u \in X_i$ et $v \in X_i$, (i.e, il existe au moins un sac contenant u et v).*
3. *Pout tout sommet v dans V, l'ensemble $\{i \in I : v \in X_i\}$ induit un sous-arbre de T.*

Remarque 43
- *Notons que la terminologie que nous utiliserons pour parler de la notion de sommetsest 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.*
- *Dans la suite, on notera aussi 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 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 ci-dessous 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 de décomposition arborescente. Cette figure présente aussi la triangulation de G pour laquelle T est un arbre de cliques.

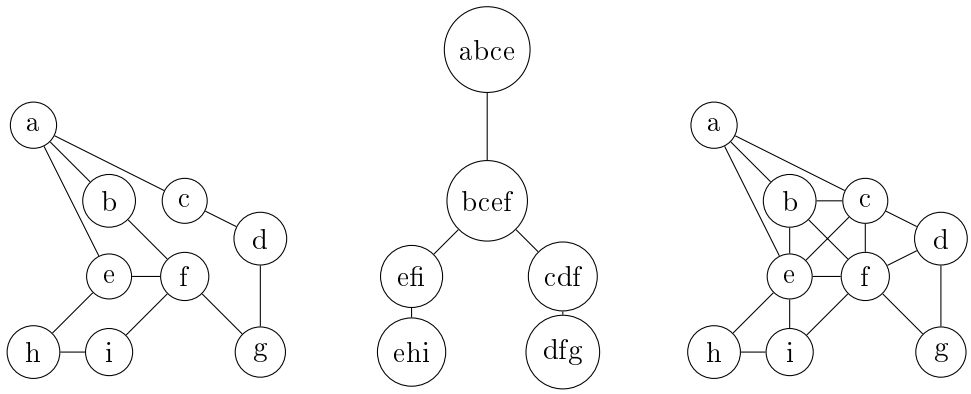


FIGURE 1.10 – Un graphe G, une décomposition arborescente T de G et la triangulation correspondante

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 sont alors trivialement respectés. La proposition suivante présente les propriétés de séparations induites par les décompositions arborescentes :

Proposition 1.9.1 *[46] Soient T une décomposition arborescente d'un graphe G et B_1, B_2 deux sacs adjacents de T. Soient T_1 et T_2 les deux arbres obtenus en supprimant l'arête (B_1, B_2) dans T. Soient $V_1 = \bigcup_{u \in V(T_1)} X$ et $V_2 = \bigcup_{u \in V(T_2)} X$. Pour toute paire de sommets u et 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 $B_1 \cap B_2$.*

Pour illustrer cette proposition, revenons à la figure précédente. 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.

1.9.1 Largeur arborescente

Définition 44 *La largeur d'une décomposition arborescente (en anglais Tree-width) est le nombre maximum de sommets contenus dans un de ses sacs moins* 1.

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

La largeur arborescente d'un graphe est la plus petite des largeurs de toutes les décompositions arborescentes possibles de G, i.e., $tw(G) = \min_T\{Largeur(T)\}$.

Dans la figure 1.10, 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 prolifique. En effet il a été constaté assez tôt que certains problèmes réputés difficiles peuvent être résolus pour des graphes de largeur arborescente bornée [2]. 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 qui peuvent être résolus en temps linéaire pour des graphes de largeur arborescente bornée, voir les travaux de Courcelle [40], Arnborg, Lagergren et Seese [2] et de Courcelle et Mosbah [39].

Malheureusement, trouver une décomposition arborescente de largeur minimum est un probl NP-complet [1], et ce même pour des classes de graphes assez restreintes comme les graphes de degré borné [17], les graphes bipartis ou les graphes de co-comparabilité [72].

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 45 *[84]*

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.

1.9.2 Longueur arborescente

Définition 46 *La longueur d'une décomposition arborescente (en anglais Tree-length) est le diamètre maximum des sous-graphes induits par les sommets contenus dans les sacs.*

$Longueur(T) = \max_{B \in V(T)}\{diam_G(B)\}$

La longueur arborescente d'un graphe est la plus petite des longueurs de toutes les décompositions arborescentes possibles de G. i.e, $tl(G) = \min_T\{Longueur(T)\}$.

1.9.3 Décomposition arborescente 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 47 *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 $B_1 \backslash B_2 \neq \emptyset$ et $B_2 \backslash B_1 \neq \emptyset$. ($V(T)$ étant l'ensemble de sacs de T).*

Proposition 1.9.2 *[50]*

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 d'une dmposition 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), T reste une décomposition arborescente de G.

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 1.9.3 *[16]*

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

Chapitre 2

Routage Compact

Sommaire

La construction de schémas de routage pour les familles de graphes admettant une décomposition arborescente dont les sacs sont de diamètre bornés, proposés par Dourisboure [50] a été le point de départ et l'une des principales motivations de nos travaux. Dans ce chapitre, nous présentons les principaux résultats obtenus dans la littérature concernant la problématique du routage compact.

2.1 Définition du routage

La problématique du routage consiste à : Étant donné un réseau de processeurs communiquants, modélisé par un graphe non orienté $G = (V, E)$ fixé à l'avance, fournir dans chaque routeur (représenté par un nœud du graphe) un algorithme de routage aussi efficace que possible.

Un algorithme de routage est une fonction calculable qui détermine le lien sur lequel le message doit être transmis pour chaque message et pour chaque lien par rapport à la destination contenue dans l'en-tête du message.

Un algorithme de routage sera dit *efficace*, s'il englobe un certain nombre de qualités comme ; Les routes générées par l'algorithme de routage sont de plus courts chemins dans le graphe, le temps de calcul de la fonction de routage est faible, le nombre de routes utilisant le même lien est faible, etc.

2.1.1 Schémas de routage

Un schéma de routage est composé de :

1. $adresse(u)$: adresse de la machine u (chaîne binaire [5]).
2. les en-têtes ajoutées à tout message : sont constituées 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).

2.1.2 Caractéristiques d'un schéma de routage

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, i.e., optimiser la complexité mémoire des schémas de routages. $|adresse(u)| + |mem(u)|$ = nombre de bits nécessaires pour coder respectivement l'adresse de la machine u et sa mémoire locale.

Longueur des routes

Étant 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, on a $\rho_R(u, v) \leq a.dist_G(u, v)$.

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 écarter des plus courts chemins lorsqu'il fait suivre un message.

5. Une chaîne binaire est une chaîne de valeurs binaire

Rapidité de décision

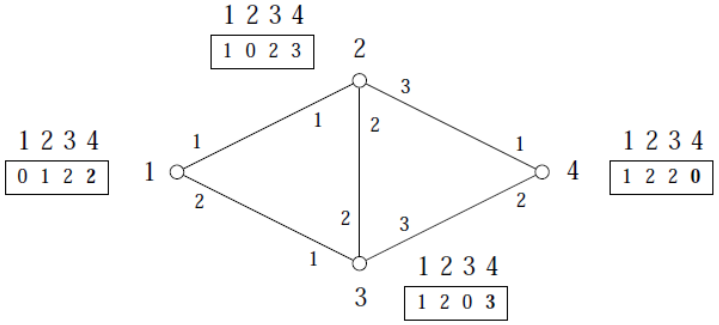
C'est le temps nécessaire à un routeur pour décider vers où il doit faire transiter le message. Ce paramètre se calcule en étudiant la complexité en temps de la fonction R_u.

2.1.3 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 $\{1, 2, ..., n\}$. Les en-têtes se limitent à l'adresse de la destination. Ainsi, en-têtes 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 $d = deg(u)$.

Illustration. Voir l'exemple suivant [63] :

Soit $G = (V, E)$ le graphe défini par $V = \{1, 2, 3, 4\}$ et $E = \{12, 13, 23, 24, 34\}$.



↱	1	2	3	4
1	0	1	2	**2**
2	1	0	2	3
3	1	2	0	**3**
4	1	2	2	**0**

FIGURE 2.1 – Table de routage

Nous proposons sur G une fonction de routage qui est représentée par la table de la figure 3.1. Au cours du routage, les en-têtes représentent la destination et ne varient pas. Le port de sortie (valeur indiquée dans la table), en chaque sommet, ne dépend que de l'adresse de destination. Cette table se lit de la manière suivante : un message de source le sommet 1 et de destination le sommet 4, utilisera le port de sortie indiqué dans la ligne 1 et la colonne 4 de la table. Ainsi, la route d'un message allant du sommet 1 au sommet 4, utilise le port de sortie 2, une fois au sommet 3 utilise le port de sortie 3, puis arrive au sommet 4 par le port d'entrée 2. Donc, dans cet exemple, nous avons $R_1(0, 4) = (2, 4)$, $R_3(1, 4) = (3, 4)$ et $R_4(2, 4) = (0, 4)$. Les fonctions locales de routage, R_1, R_2, R_3, R_4 (ligne de la table) sont reportées près de chaque sommet. Cette fonction de routage est sommet-préservante car $L(u) = ID(u)$ pour tout sommet u.

Une fonction de routage qui, en chaque sommet, ne dépend que de l'en-tête et non du port d'entrée est dite "oblivious" (oubliante). Et, une fonction de routage oblivious qui dépend seulement de la destination (c'est-à-dire telle que $h_0 = h_1 = ... = h_r = L(v)$, est une fonction de *routage directe* ou encore *table de routage*.

Toutes ces distinctions ont un impact sur la mise en œuvre des fonctions de routage. La modification des en-têtes par exemple, peut être coûteuse pour les réseaux de type «tout optique» impliquant des conversions optique-électrique. Les fonctions de routage directes peuvent être codées simplement au moyen d'une table, comme dans l'exemple précédent. Elles ont aussi la propriété d'être sans boucles, c'est-à-dire que la route traversée par un message ne peut pas comporter de cycles, puisque sinon, le message bouclerait indéfiniment sans atteindre sa destination, contredisant la définition même d'une fonction de routage (existance d'un chemin entre toute paire de sommets).

2.2 Routage compact

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

2.2.1 Routage compact dans les arbres

Les schémas de routage selon les plus courts chemins dans les arbres sont du à Fraigniaud et Gavoille dans [58] et à Thorup et Zwick dans [126].

Théorème 48 *[58]*
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.*

2.2.2 Routage compact dans les graphes triangulés

Les schémas de routage dans les graphes triangulés sont du, aux travaux de Dourisboure et Gavoille [49, 51, 52].

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.

Avec ces deux arbres T et S, Dourisboure [50] a défini la stratégie de routage dans le graphe G suivante : 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 2.2.1 *[50]*
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 $v \in B(u)$.

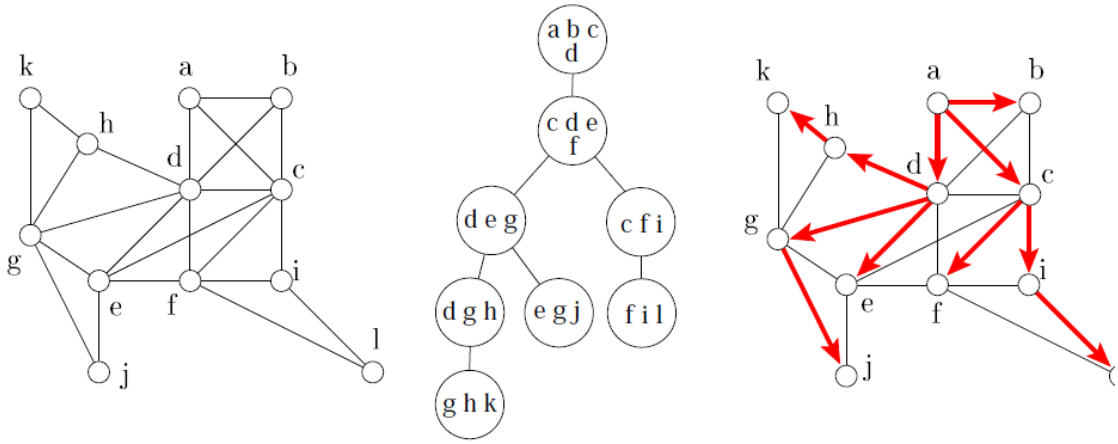


FIGURE 2.2 – Un graphe triangulé G, un arbre de cliques T et un arbre S couvrant G

Proposition 2.2.2 *[50]*
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 49 *[51]*
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 $O(\log k)$.

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. Étant 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 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 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 $X = nca_H(B(u), B(v))$ avec $(nca_H(B(u), B(v))$ signifie : le plus petit ancêtre commun entre $B(u)$ et $B(v)$ dans l'arbre hiérarchique H).

Définition 50 *Il est connu que tout arbre à n nœuds possède un médian, c'est-à-dire un sommet s tel que $T - \{s\}$ est une forêt 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 $T - \{s\}$. Notons que H et T ont le même ensemble de nœuds mais H est de profondeur au plus $\log n$.

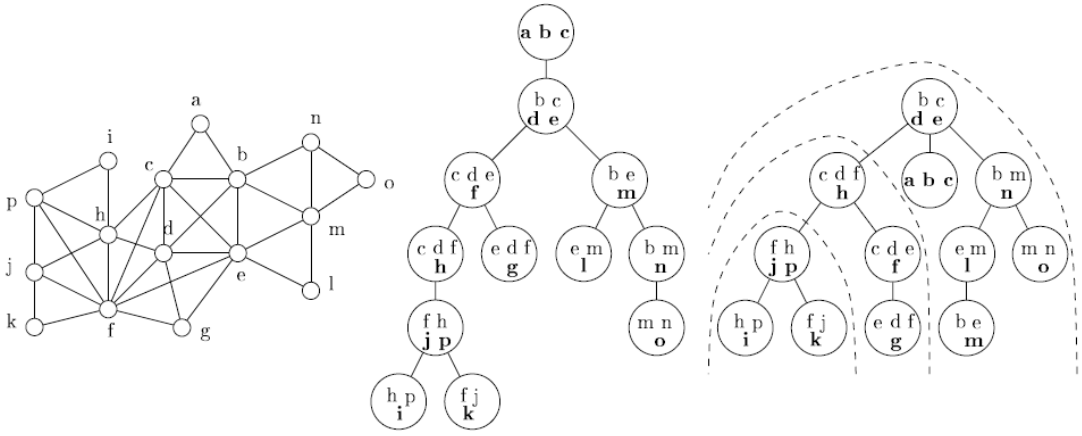


FIGURE 2.3 – Un graphe triangulé G, un arbre de cliques T et un arbre hiérarchique H

On aboutit au théorème suivant :

Théorème 51 *[50]*
Tout graphe triangulé admet un schéma de routage de déviation 2, sans boucles, indépendant 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.*

Schéma de routage de déviation 1

Yon Dourisboure a proposé un schéma de routage de déviation 1, qui reprend le schéma de routage de Peleg [114] qui route selon les plus courts chemins, mais en compressant la taille des adresses et en s'autorisant une déviation de 1.

Théorème 52 *[49]*
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.*

2.2.3 Routage compact dans les graphes de longueur arborescente bornée

Yon Dourisboure et cyril Gavoille ont adapté un schéma aux graphes qui n'admettent pas 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 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 $X = nca_H(B(u), B(v))$. 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 53 *[50]*
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.

2.2.4 Autres résultats sur le routage compact dans les graphes

Dans cette partie, nous allons présenter très brièvement les principaux résultats sur le routage compact dans les graphes contenus dans la thèse de Dieng [47].

Dans un premier temps, nous allons nous intéresser au routage dans les graphes planaires-extérieurs. On a le résultat suivant :

Théorème 54 *[47]*
Tout graphe planaire-extérieur connexe et non valué à n sommets admet un schéma de routage de plus courts chemins avec des adresses, des entêtes et des tables de routage $O(\log n)$ bits. De plus le temps de routage est constant.

Ce résultat est une généralisation du schéma de routage pour les arbres, présenté par Thorup et Zwick [126] et indépendamment par Fraigniaud et Gavoille [58].

Dans un premier temps, l'auteur donne une généralisation aux graphes valués (k, r)-cellulaires. Les graphes (k, r)-cellulaires sont des graphes ne contenant pas $K_{2,r}$ comme mineur et dont les composantes 2-connexes peuvent être rendues planaire-extérieures par la suppression de k sommets. Par exemple, les réseaux $(2, 4)$-cellulaires contiennent les arbres, les graphes planaire-extérieurs et plus généralement tous les graphes sans mineur $K_{2,4}$ qui contiennent, par exemple des arbres de K_5 ou de $K_{3,3}$ qui sont de genre non borné. Plus précisément, l'auteur a montré le résultat suivant :

Théorème 55 *[47]*
Tout graphe (k, r)-cellulaire connexe, valué à n sommets admet un schéma de routage de plus court chemins avec des adresses et des en-têtes de $O(\log r \log n)$ bits et des tables de routage de $O(kr \log n)$ bits. Le temps de décision est de $O(\log r)$.

Puisque les graphes sans mineurs $K_{2,4}$ sont $(2, 4)$-cellulaires, alors on a le corollaire suivant :

Corollaire 2.2.1 *[47]*
Tout graphe sans mineur $K_{2,4}$, connexe et valué à n sommets, admet un schéma de routage de plus courts chemins avec des tables, des adresses et des en-têtes de message de taille $O(\log n)$ bits. Le temps de routage est constant.

Dans un deuxième temps, Dieng-Youssou a étendu les résultats sur routage dans les graphes planaire- extérieurs à une autre famille de graphes plus générale, les graphes t-feuillets. Un graphe t-feuillet G est un graphe dont les arêtes peuvent être partitionnées en un ensemble de sous-graphes de G planaire-extérieurs tels que chaque arête de G soit contenue dans au moins un sous-graphe et il y a au plus t sommets appartenant à la fois à plusieurs sous-graphes. Par exemple, les graphes planaire-extérieurs sont 0-feuillet. Plus précisément, on a le résultat suivant :

Théorème 56 *[47]*

Tout graphe t-feuillet connexe, non valué possède un schéma de routage de plus courts chemins utilisant des tables de $O(t \log n)$ bits par sommets, des adresses et des en-têtes de $O(\log n)$ bits. Le temps de décision est de $O(\log t)$ par sommet.

Chapitre 3

Techniques de décompositions de graphes

Sommaire

Après avoir donné les définitions principales concernant les graphes dans le premier chapitre, les motivations concernant la problématique du routage dans le deuxième chapitre, nous allons dans ce troisième chapitre, présenter les différentes méthodes de calcul des décompositions arborescentes des graphes. Ensuite, nous présenterons les invariants liés à ce concept et nous nous focaliserons sur les deux invariants étroitement liés entre eux ; la largeur arborescente et la longueur arborescente. Nous donnerons les principaux résultats relatifs au calcul de ces deux paramètres sur les classes de graphes, et notamment notre première contribution, il s'agit du calcul de la longueur arborescente pour les graphes faiblement triangulés et k-cordaux, travaux au cœur de notre mémoire de magister [21], une version rapport de recherche a par ailleurs été publiée dans les actes de la conférence COSI'06 [22] et une autre version actualisée soumise e revue internationale [29].

Les techniques de décomposition de graphes (comme par exemple la décomposition modulaire, la split décomposition, la décomposition arborescente, etc) relèvent toutes d'un même principe, connu sous le nom de diviser pour régner. L'idée est de décomposer une (grande) structure en (petites) sous-structures, sur lesquelles il sera plus facile de résoudre un problème donné. L'intérêt de ces techniques de décomposition réside dans le fait que, pour certains problèmes qui

sont difficiles en général, il est possible de les résoudre efficacement lorsque les graphes concernés admettent une "bonne" décomposition.

3.1 Méthodes de calcul des décompositions arborescentes

Dans sa thèse, Ndiaye [107] a présenté un état de l'art très détaillé sur les classes d'algorithmes proposés pour le problème de calcul des décompositions arborescentes des graphes, dont nous proposons un résumé non exhaustif dans cette section.

Il existe un lien très étroit entre la notion de triangulation et celle de décomposition arborescente.

Le moyen le plus simple pour calculer une décomposition arborescente pour un graphe donné est de calculer une triangulation de ce graphe.

Théorème 57 *[116]*
Soient G un graphe et $k \in \mathbb{N}^+$. La largeur arborescente de G est au plus $k-1$ si et seulement si G admet une triangulation H dont la taille maximale des cliques est bornée par k.

Par ailleurs, la largeur arborescente d'un graphe triangulé est égale à la taille maximum de ses cliques moins un, et on a aussi tout graphe triangulé admet une décomposition arborescente dont les sacs sont des cliques maximales.

Donc, pour un graphe quelconque G, le calcul d'une triangulation implique une décomposition arborescente de G qui correspond exactement à l'arbre de cliques de ladite triangulation.

On note également qu'un graphe triangulé G a au plus n cliques maximales [61] qui peuvent être calculées en un temps linéaire [65] où n est le nombre de sommets de G.

3.1.1 Les méthodes exactes

Le problème du calcul d'une triangulation optimale étant NP-complet, il n'existe pas d'algorithme polynomial qui calcule cette triangulation optimale. Les méthodes proposées sont donc de complexité exponentielle, dont l'intérêt pratique est très limité, eu égard au temps de calcul qui n'est point raisonnable.

Dans [121], Shoiket et Geiger (1997), proposent l'algorithme **QuickTree**, qui selon eux est le premier à pouvoir calculer une triangulation optimale en un temps raisonnable. Cet algorithme est basé sur la notion de séparateurs minimaux. Il se dote d'une valeur k qui est un minorant de la largeur arborescente du graphe et calcule l'ensemble des séparateurs minimaux $S(k)$ dont la taille est au plus k, appelé ensemble k-régulier. A chaque séparateur est associé un ensemble de fragments qui sont les unions entre les composantes connexes induites par les séparateurs et les séparateurs saturés (transformés en cliques). Si pour un séparateur de $S(k)$, les fragments associés sont minimalement k-triangulés (cliques maximales de taille au plus k) alors la composition de ces derniers donne une triangulation minimale du graphe de départ de largeur au plus k. Sinon, les fragments sont ordonnés de manière croissante suivant leur taille. Pour chaque fragment non triangulé, QuickTree calcule si possible, un ensemble k-régulier de séparateurs minimaux de cet ensemble tel que chaque fragment est soit une clique, soit minimalement k-triangulé. La composition de ces différents fragments triangulés donne une triangulation minimale de largeur arborescente au plus k et de ce fait une triangulation optimale. Si un ensemble vérifiant ces conditions n'existe pas, cela veut dire que la largeur arborescente de G est supérieure à k.

Dans [67], Gogate et Dechter (2004), ont comparé QuickTree à la méthode **QuickBB**, cette denière calcule la largeur arborescente par une technique de Branch and Bound sur l'ensemble des ordres des sommets du graphe. On commence par calculer un majorant et un minorant grâce à des techniques heuristiques. Les auteurs proposent une heuristique de calcul de minorant appelée *minor-min-width*. Le minorant de départ est $lb = 0$. À chaque étape, l'heuristique contracte l'arête entre un sommet de degré minimum v dans le graphe courant et un sommet de degré minimum dans son voisinage. Ensuite, elle met à jour lb qui prend la valeur maximum entre sa valeur actuelle et le degré de v avant la contraction. Ce traitement est poursuivi jusqu'à ce qu'il n'y ait plus aucun sommet dans le graphe. Les expérimentations effectuées dans l'article montrent que minor-min-width donne de bons résultats.

Si, le majorant et le minorant de départ sont égaux alors la largeur arborescente est égale à cette valeur. Sinon, QuickBB réalise une recherche par branch and bound avec une construction incrémentale de schémas d'élimination parfaits sur les sommets, en maintenant un minorant sur la largeur de la triangulation en cours de construction. Le majorant au début du branch and bound est donné par le majorant fourni par l'heuristique choisie. À chaque étape, un sommet est éliminé. Cela engendre la complétion de son voisinage dans le graphe courant et une mise à jour du minorant. Ce dernier aura pour valeur le maximum entre son ancienne valeur et le degré du sommet éliminé. L'heuristique minor-min-width calcule un minorant de la largeur du graphe courant qui est agrégée au minorant courant. Si le minorant dépasse le majorant, un backtrack est réalisé pour choisir un autre sommet. Sinon, QuickBB choisit un nouveau sommet dans le graphe courant. Si ce graphe est vide, le majorant est mis à jour et sa nouvelle valeur est la largeur de la triangulation construite. La recherche continue jusqu'à ce que la totalité de l'espace des ordres possibles soit visitée. La largeur arborescente est donc la dernière valeur du majorant. La méthode peut être améliorée grâce à des techniques de réduction du graphe. En effet, certains sommets du graphe courant peuvent être éliminés sans affecter la procédure de branch and bound. Les règles du sommet simplicial ou quasi-simplical [14], permettent d'éliminer tout sommet (quasi-)simplicial dans le graphe courant. Dans le cas d'un sommet simplicial, la valeur du minorant devient le maximum entre son ancienne valeur et le degré de ce sommet parce que la largeur de cet ordre sera au moins égale au degré du sommet. On peut de même éliminer un sommet quasi-simplicial dont le degré est inférieur à la valeur du minorant car cela n'augmente pas la largeur de la triangulation en cours de construction. En outre, le choix du prochain sommet peut être restreint à ceux qui ne sont pas adjacents au sommet courant. D'autres résultats théoriques permettent aussi de rajouter directement une arête entre deux sommets non encore considérés, de faire de la propagation et un élagage plus important.

Grâce à ces techniques, QuickBB s'avère assez efficace, même si sa complexité est exponentielle $O(n^{n-tw})$, où tw est la largeur arborescente du graphe. La comparaison réalisée entre QuickBB et QuickTree donne des résulats nettement en la faveur du premier. Cela est vérifié sur des graphes structurés (largeur arborescente limitée) mais surtout sur ceux sans structure particulière grâce à une complexité qui diminue avec l'augmentation de la largeur arborescente. Or, sur ces graphes non structurés, le nombre de séparateurs minimaux peut être exponentiel et rend donc QuickTree inopérationnel. QuickBB part également avec un avantage non négligeable grâce au majorant calculé au départ, avec l'heuristique minfill. Nous allons voir dans la suite que minfill est d'une très grande efficacité dans le calcul d'un majorant de la largeur arborescente de qualité.

Dans [20], Bouchitté et Todinca (2002), donnent une classe de graphes pour laquelle le problème de calcul de la largeur arborescente est polynomial. Ils résolvent la conjecture $ESA'93$ qui

disait que les problèmes de largeur arborescente et de minimum fill-in [6] sont polynomiaux pour les graphes ayant un nombre polynomial de séparateurs minimaux. Ils proposent dans ce cadre un algorithme de triangulation basé sur la notion de cliques maximales potentielles.

Définition 58 *Un ensemble de sommets S de G est une clique maximale potentielle si S est une clique maximale dans une triangulation minimale de G.*

Bouchitté et Todinca (2001) [19] avaient déjà montré que les problèmes de calcul de la largeur arborescente et du minimum fill-in étaient polynomiaux pour les graphes dont l'ensemble des cliques maximales potentielles pouvait être énuméré en un temps polynomial. De même, qu'il était possible d'énumérer polynomialement l'ensembles des cliques maximales potentielles si on dispose de l'ensemble des séparateurs minimaux du graphe. Cela donne le résultat suivant :

Théorème 59 *[19] La largeur arborescente et le minimum fill-in d'un graphe peuvent être calculés en un temps polynomial en la taille du graphe et le nombre de ses séparateurs minimaux. La complexité de l'algorithme est en $O(n^3 N b^3_{sepmin} + n^2 m N b^2_{sepmin})$.*

Ils utilisent la programmation dynamique à l'instar de [121] pour résoudre ces problèmes à partir de l'ensemble des séparateurs minimaux et des cliques maximales potentielles.

3.1.2 Les méthodes heuristiques

Ces approches construisent en général un ordre (dynamiquement) et ajoutent des arêtes dans le graphe, de sorte qu'en fin de traitement, l'ordre obtenu soit un ordre d'élimination parfait pour le graphe résultant G'. La complexité de ces méthodes est en général polynomiale (souvent même linéaire) mais elles n'offrent, en contrepartie, aucune garantie d'optimalité. D'après [92], cette approche est justifiée en pratique. En effet, Kjaerulff a observé entre autres, qu'au contraire des résultats attendus, ces heuristiques produisent des triangulations raisonnablement proches de l'optimum. De plus, elles sont en général faciles à implémenter. Nous présentons ci-dessous les plus célèbres et souvent les plus efficaces, dont la complexité en temps varie de $O(n + m')$ à $O(n(n + m'))$, m' étant le nombre d'arêtes de G'.

- **LBFS** [118]. Elle a pour objet la reconnaissance des graphes triangulés. Elle ordonne les sommets de n à 1 en étiquetant tous les sommets à vide au départ et en choisissant arbitrairement le premier sommet (numéroté n). Elle rajoute n à l'étiquette des sommets voisins du sommet n. Ensuite, elle choisit comme sommet suivant un sommet ayant la plus grande étiquette suivant un ordre lexicographique et rajoute le numéro du sommet aux étiquettes de ses voisins non encore numérotés (en cas d'égalité, LBFS fait un choix arbitraire). Sa complexité est en $O(n + m')$.
- **MCS** [125]. Elle consitue un "cousin simplifié de LBFS". C'est un algorithme de reconnaissance des graphes triangulés. Elle ordonne les sommets de n à 1 en choisissant arbitrairement le premier sommet (numéroté n) et ensuite comme sommet suivant un sommet ayant le plus grand nombre de voisins déjà numérotés (en cas d'égalité, MCS fait un choix arbitraire). Sa complexité est en $O(n + m')$.
- **Mindeg-intérieur**. Elle ordonne les sommets de 1 à n, en sélectionnant comme nouveau sommet, le sommet qui possède le plus petit nombre de voisins déjà numérotés. Sa complexité est généralement en $O(n^3)$.

6. Le problème du minimum fill-in consiste à rechercher le surgraphe triangulé qui a le plus petit nombre d'arêtes possible

- **Mindeg**. Elle ordonne les sommets de 1 à n, en sélectionnant comme nouveau sommet, le sommet qui possède le plus petit nombre de voisins non numérotés. Sa complexité est généralement en $O(n^3)$.
- **Minfill**. Elle ordonne les sommets de 1 à n, en sélectionnant comme nouveau sommet, le sommet qui conduira à ajouter un minimum d'arêtes si l'on complète le sous-graphe induit par ses voisins non encore numérotés. Sa complexité est en $O(n(n+m^{'}))$.

D'autres méthodes heuristiques existent dans la littératures, qui calculent des décpompositions arborescentes suivant un critères de longueur arborescente au lieu de la largeur arborescente :

Une décomposition en temps linéaire avec l'algorithme BFS - Layering.

L'algorithme BFS-Layering présenté dans la thèse de Dourisboure [50] est une adaptation d'un algorithme de Chepoi, Dragan et Yan dans [36], qui construit en temps linéaire $O(nk+m)$ un sous graphe couvrant $(1, k+1)$-approximant avec $2n-2$ arêtes pour tout graphe k-cordal.

Définition 60 *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 : $B^i = \bigcup_{j=0}^{i} C_j^i$

Un partitionnement en couches de G est un partitionnement de chaque C^i en $C_l^i, ..., C_{p_i}^i$ 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 3.1.1 *[50]*
H est calculable en temps linéaire $O(n+m)$ et H est un arbre. H est appelé, un arbre de partitionnement en couches.

Théorème 61 *[52]*
L'algorithme BFS-Layering calcule en temps $O(n+m)$ une décomposition arborescente T d'un graphe G telle que :

- *$longueur(T) \leq k/2 + 3$ où k est la cordalité de G,*
- *$longueur(T) \leq 3.tl(G) + 1$.*

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

- 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 62 *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*

$$border(T) = \{u \in covered(T) / N(u) - covered(T) \neq \emptyset\}$$

.

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 :

$$Cost(S, u) = \begin{cases} \infty \text{ si } u \in border(T) \\ 0 \text{ s'il n'existe pas de sommets en conflits dans } S - \{u\} \\ \max_{x,y \in S} dist_G(x, y) \text{ tels que } x, y \text{ sont en conflits dans } S - \{u\} \text{ sinon} \end{cases}$$

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.

Théorème 63 *[52]*
L'heuristique se termine en temps polynomial. De plus lorsqu'elle se termine avec succès, T est une décomposition arborescente de G de longueur au plus $2k$.

3.1.3 Les algorithmes minimaux

A l'image des algorithmes heuristiques, les algorithmes minimaux calculent des triangulations avec un coût assez faible. Mais contrairement aux heuristiques, ils nous garantissent que le retrait d'une arête rajoutée pour la triangulation d'un graphe donne un nouveau graphe qui n'est pas triangulé. En effet, une triangulation minimale ne contient pas d'arête redondante.

Une triangulation peut être minimale sans être optimale. Mais il est évident que, si on rajoute des arêtes à une triangulation d'un graphe, on ne peut que s'éloigner de l'optimum. En outre, l'intérêt de ce type d'approche repose sur l'existence d'algorithmes de complexité polynomiale. Ces derniers proposent une approximation de l'optimum sans garanties autres que toutes les arêtes présentes soient nécessaires. Nous présentons ci-dessous, un certain nombre d'algorithmes de triangulation minimale. Pour plus de détails, [Heg06] propose un survey très complet sur ces méthodes.

- **Lex-M** [118]. C'est une extension de LBFS qui calcule des ordres de triangulation minimale. Elle ordonne les sommets de n à 1 en étiquetant tous les sommets à vide au départ et choisissant arbitrairement le premier sommet v (numéroté n). Elle rajoute n à l'étiquette des voisins de v et de tout sommet non numeroté u pour lequel il existe un chemin entre u et v, composé uniquement de sommets non numérotés et dont les étiquettes sont lexicographiquement plus petites que celles de u et v. Ces chemins sont appelés des fill-paths. Ensuite, elle choisit comme sommet suivant un sommet v ayant la plus grande étiquette suivant un ordre lexicographique et rajoute le numéro du sommet à l'étiquette des voisins de v et de tout sommet non numeroté u pour lequel il existe un fill-path entre u et v (les égalités sont cassées arbitrairement). Sa complexité est en $O(nm')$.
- **MCS-M** [8, 9] C'est une extension de MCS qui calcule des ordres de triangulation minimale. Elle consitue donc un "cousin simplifié de Lex-M". Elle ordonne les sommets de n à 1 en donnant un poids nul à tous les sommets au départ et choisissant arbitrairement le premier sommet v (numéroté n). Elle incrémente le poids des voisins de v et de tout sommet non numeroté u pour lequel il existe un chemin entre u et v, composé uniquement de sommets non numérotés dont les poids sont plus petits que ceux de u et v. Ces chemins sont des fill-paths. Ensuite, elle choisit comme sommet suivant un sommet v ayant le plus grand poids et incrémente le poids des voisins de v et de tout sommet non numeroté u

pour lequel il existe un fill-path entre u et v (les égalités sont cassées arbitrairement). Sa complexité est en $O(nm^{'})$.

- **LB-Triang** [12]. Cet algorithme utilise l'algorithme LB-Triang (voir section 3.3.4) pour calculer une triangulation minimale. Il utilise un ordre quelconque sur les sommets et les rend LB-simpliciaux suivant cet ordre. Il peut calculer toutes les triangulations minimales possibles, suivant l'ordre utilisé. Le prochain sommet x est choisi suivant l'ordre puis les séparateurs minimaux inclus dans le voisinage de x sont calculés puis complétés. Compléter les voisinages des composantes connexes induites par les voisinages des sommets du graphe avait déjà été proposée par Fujisawa et Orino [60] et présentée dans [93] avec une complexité en $O(n(m+m^{''}))$, $m^{''}$ étant le nombre d'arêtes rajoutées. Là où les méthodes classiques complètent tout le voisinage, cet algorithme complète la partie nécessaire avec une complexité en $O(nm^{'})$.

Ces trois algorithmes construisent des sommets LB-simpliciaux suivant un certain ordre. Il existe d'autres méthodes qui pour calculer une triangulation minimale partent d'une triangulation quelconque et retirent des arêtes jusqu'à avoir une triangulation minimale. Cette approche était justifiée par le fait que, jusqu'à la définition de LB-Triang, LEX-M était le meilleur algorithme de triangulation avec une complexité en $O(nm^{'})$.

3.2 Largeur arborescente des graphes

Dans cette section, nous allons approfondir la notion de largeur arborescente, qui généralise la propriété d'avoir de petits séparateurs. Intuitivement, un graphe est de largeur arborescente petite s'il peut être récursivement décomposé en petits sous-graphes se chevauchant légèrement, ou encore plus intuitivement, si le graphe ressemble à un arbre. Beaucoups de problèmes qui sont NP-difficile pour les graphes quelconques peuvent être résolus en temps polynomial pour les graphes de petite largeur arborescente.

Nous allons commencer par donner une caractérisation des graphes de largeur arborescente k. Ensuite nous donnerons un exemple illustratif [41] pour le problème du stable qui est connu pour être NP-Complet en général mais devient polynomial pour la classe des graphes de largeur arborescente assez petite.

Nous allons décrire une technique d'un schéma d'approximation en temps polynomial pour le problème du stable maximum.

Un graphe G est dit k-arbre si et seulement si G est le graphe complet avec k sommets, ou G possède un sommet v de degré $k-1$ tel que $G-v$ est un k-arbre. Un k-arbre partiel est sous graphe d'un k-arbre. En particulier, un graphe connexe G est un 1-arbre si et seulement si G est un arbre. Par recursivité on aura un ordre d'élimination $v_1, v_2, ..., v_n$ pour les sommets de tous les k-arbres partiels, dans lesquels chaque sommet v_i a au plus $k-1$ voisins de plus grand indice.

On peut facilement déduire une décomposition arborescente de largeur k pour tout k-arbre partiel à partir de l'ordre d'élimination. Plus précisément, la décomposition arborescente T a n nœuds $\{1, 2, ..., n\}$, et pour tout nœud i, le sous ensemble $X(i)$ contient le sommet v_i et ces (au plus $k-1$) voisins d'indices plus grand. La contraposée est aussi vraie ; toute décomposition arborescente de largeur k, définit un ordre d'élimination prouvant que le graphe décomposé est un k-arbre partiel.

Théorème 64 *[130, 119]*
Un graphe est de largeur arborescente k si et seulement si il est un k-arbre partiel.

Finalement, une décomposition arborescente (T, X) peut être interprétée comme une hiérarchie récursive de séparateurs comme suit : Choisir un nœud arbitraire de T pour être la racine. Pour chaque nœud i de T, soit $D(i)$ l'union des sous ensembles $X(j)$ pour tous les descendants de j de i (incluant i), et soit $A(i)$ l'union des sous ensembles $X(j)$ pour tous les ancêtres propres de j de i. Chaque nœud i est associé avec un ensemble de sommets limites $B_i = D(i) \cap A(i)$ et un sous graphe G_i, qui est le sous graphe induit $G[D(i)]$ moins les arêtes entre les nœuds limites. Chaque sous ensemble $X(i)$ est un séparateur du sous graphe correspondant G_i.

On peut relier la largeur arborescente des graphes à l'existance des séparateurs comme suit. On dira qu'un graphe à n-sommets G est $s-$séparable, pour une fonction non-décroissante

$$s : \mathbb{N} \to \mathbb{N},$$

si on a soit $n = 1$ ou G a un 2,3-séparateur S de taille $s(n)$, tel que les composantes connexes de $G - S$ sont aussi s-séparable.

Théorème 65 *[130, 119] Tout graphe de largeur arborescente k est $O(k)$-séparable. Réciproquement, tout graphe s-séparable est de largeur arborescente $O(s(n)\log n)$, ou de largeur arborescente $O(s(n))$ si $s(n) = \Omega(n^c)$pour une constante $c > 0$.*

Corollaire 3.2.1 *Tout graphe planaire est de largeur arborescente $O(\sqrt{n})$.*

Programmation dynamique : Stable maximum

Un ensemble stable dans un graphe G est un sous ensemble de sommets, tel qu'aucune paire n'est reliée par une arête dans G. Calculer le plus grand stable dans un graphe est l'un des problèmes NP-difficiles classiques. De plus, même l'approximation du plus grand stable avec un facteur de $n^{1-\epsilon}$, pour tout $\epsilon > 0$, est NP-difficile [71, 132].

Néanmoins, pour tout graphe de largeur arborescente constante (ou logarithmique), on peut calculer le plus grand stable en temps polynomial.

Théorème 66 *[3]*
La taille d'un stable maximum dans un graphe de largeur arborescente k peut être calculé en temps $O(4^k k^2 n)$.

Dans ce qui suit, nous donnons la preuve de ce théorème, afin de voir un exemple sur le calcul d'un stable maximum en temps polynomial, pour un graphe de largeur arborescente bornée par une constante.

Preuve
Soit G un graphe de largeur arborescente k, et soit (T, X) une décomposition arborescente de largeur k. Sans perte de généralité, l'arbre T a au plus n nœuds. Soit r une racine de T choisie arbitrairement. Pour chaque nœud i, soit G_i et B_i le sous graphe et sommets limites correspondants (comme définis plus haut). Pour tout nœud i de T et tout sous ensemble $A \subseteq B_i$, Soit $MIS(i, A)$ la taille du plus grand stable I dans G_i tel que $I \cup B_i = A$. Cette fonction obeillit à la récurrence :

$$MIS(i, A) = \{|A| + \max\{\sum_{j fils\ de i} MIS(j, B_j \cup A') - |B_j \cup A|\}$$

tel que J est un stable dans G_j et $A \subseteq J \subseteq X(i)\}$. Avec le cas basique $MIS(i, A) = |A|$ si $G_i - B_i = \emptyset$ (en particulier, si i est une feuille de T). Le stable maximum dans G est de taille

$MIS(G, \emptyset)$. On peut facilement évaluer la récurrence en utilisant la programmation dynamique. Pour chaque nœud i, il existe au plus 2^{k+1} sous ensembles A à considérer. Pour chaque sous problème (i, A), et chaque enfant j de i, il existe au plus 2^{k+1} sous ensembles J à considérer ; pour chaque sous ensemble J, on peut facilement déterminer en temps $O(k^2)$ si J est un stable dans G_j. Ainsi, le temps total pour évaluer $MIS(i, A)$, sans compter la recursion, est $O(2^k k^2 deg(i))$, où $deg(i)$ est le nombre d'enfants du nœud i. Ainsi, le temps de calcul total de l'algorithme de programmation dynamique est $O(4^k k^2 deg(i)) = O(4^k k^2 n)$.

□

Des stratégies similaires s'appliquent à bon nombre d'autres problèmes d'optimisation combinatoire NP-difficiles [4] ; Quelques uns sont présentés dans un survey par Arnborg et Proskurowski [3] et par Bodlaender [15].

3.3 Longueur arborescente des graphes

Dans cette partie nous faisons une étude approfondie du concept de longueur arborescente.

On note que beaucoup de classes de graphes de largeur arborescente non bornée ont une longueur arborescente bornée comme par exemple, les graphes triangulés, d'interval, scindés, sans astéroïdes triples (AT-free) et de permutation [52].

D. Lokshtanov a montré dans [104] que le problème de reconnaissance des graphes de longueur arborescente bornée par une constant $k \geq 2$ fixé est NP-complet.

Nous donnons dans ce qui suit la longueur arborescente des classes de graphes connues.

3.3.1 Longueur arborescente des graphes planaires extérieures

Un graphe planaire est un graphe qu'on peut 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 dans la Figure 3.1 l'est également.

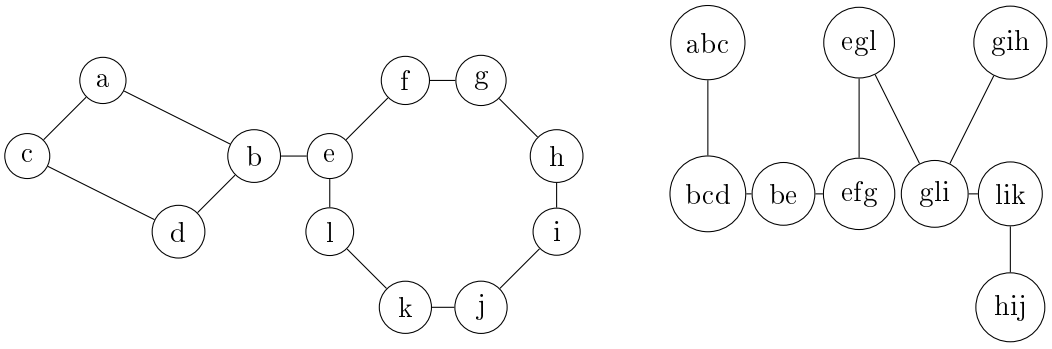


FIGURE 3.1 – Un exemple de graphe planaire extérieur et sa décomposition arborescente de longueur $\lceil k/3 \rceil$.

Théorème 67 *[52]*

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.

3.3.2 Les grilles $p \times q$

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

Définition 68 *La grille à p lignes et q colonnes, notée $M_{p,q}$, est le graphe défini par :*
- $V(M_{p,q}) = \{(i,j)/i \in \{1,...,p\}$ *et* $j \in \{1,...,q\}\}$.
- $E(M_{p,q}) = \{((x,y),(x',y'))/|x-x'| + |y-y'| = 1\}$.

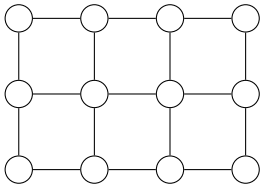


FIGURE 3.2 – Un exemple de la grille à 3 lignes et 4 colonnes

Théorème 69 *[52]*
La largeur arborescente de la grille $p \times q$ est : $tw(M_{p,q}) = \min\{p,q\}$.

Théorème 70 *[52]*
La longueur arborescente de la grille $p \times q$ est :

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

Dans ce qui suit, nous allons présenter les résultats obtenus pour la classe des graphes faiblement triangulés.

3.4 Calcul de la longueur arborescente pour les graphes faiblement triangulés et les graphes k-cordaux

Dans cette section, nous allons présenter notre première contribution dans cette thèse. Cela concerne le calcul de la longueur arborescente. On commence par calculer la longueur arborescente pour les graphes faiblement triangulés. Ensuite, nous montrons la relation qui existe entre les longueurs arborescentes des graphes et leurs triangulations minimales. On terminera par donner une borne supérieure pour la longueur arborescente des graphes k-cordaux qui est une classe de graphes assez large.

3.4.1 Les graphes faiblement triangulés

Dans cette partie, on montre que l'on peut trouver une décomposition arborescente de longueur arborescente au plus 2 pour les graphes faiblement triangulés.

Définition 71 *Les graphes faiblement triangulés sont les graphes où ni G ni son complémentaire $\overline{G}$ ne possède de cycle sans cordes induit de longueur cinq ou plus.*

Cette classe contient les triangulés, les graphes HHD - free, les graphes bipartis cordaux et les graphes distances héréditaires.

Définition 72 *Une paire de sommets $\{x, y\}$ de G est dite une 2 - paire si l'ensemble de leurs voisins communs $N(x) \cap N(y)$ forment un xy - séparateur .*

Le théorème suivant a été prouvé dans [75].

Théorème 73 *[75]*
Si G est un graphe faiblement triangulé, alors tout sous graphe induit de G différent d'une clique contient une 2 - paire.

Définition 74 *Un ensemble de sommets Ω d'un graphe G est dit clique maximale potentielle s'il existe une triangulation minimale H de G telle que Ω est une clique maximale de H.*

Bouchitté et Todinca ont donné un algorithme dans [127, 19] qui calcule la largeur arborescente des graphes ayant un nombre polynomial de séparateurs minimaux. Ils ont montré que les graphes faiblement triangulés ont un nombre polynomial de séparateurs minimaux et que cet algorithme calcule les décompositions arborescentes de largeur minimum. Dans cette décomposition arborescente, les nœuds (sacs) de celle-ci sont des cliques maximales potentielles et sont de la forme suivante $(N(x) \cap N(y)) \cup \{x\}$ ou $(N(x) \cap N(y)) \cup \{y\}$ avec $\{x, y\}$ une 2 - paire.

Proposition 3.4.1 *[22]*
La décomposition arborescente de largeur minimum pour les graphes faiblement triangulés obtenus par l'algorithme de Bouchitté et Todinca est de longueur au plus 2.

Preuve
Soit G un graphe faiblement triangulé qui n'est pas triangulé (on sait que si G est triangulé alors $tl(G) = 1$). Supposons maintenant que $|V| \geq 4$; et $tl(G) \geq 3$, ainsi $tl(G) = \min_T \{length(T)\} \geq 3$, alors $\forall T, length(T) \geq 3$, donc il existe $u, v \in \Omega$ tel que $dist_G(u, v) \geq 3$, $\Omega \in V(T)$. Supposons sans perte de généralité que $dist_G(u, v) = 3$. On sait que Ω est de la forme $(N(x) \cap N(y)) \cup \{x\}$ ou $(N(x) \cap N(y)) \cup \{y\}$ avec $\{x, y\}$ une 2 - paire. Le même raisonnement pourra être appliqué aux deux ensembles. Raisonnons sur $(N(x) \cap N(y)) \cup \{x\}$:

- *Si $x = u$ et $v \in N(x) \cap N(y)$, on a $dist_G(u, v) = 1$ (contradiction).*
- *Si $x = v$ et $u \in N(x) \cap N(y)$, on a $dist_G(u, v) = 1$ (contradiction).*
- *Si $x \neq u$ et $x \neq u$ alors $u, v \in N(x) \cap N(y)$. Ainsi u et v sont adjacents à x et on a $dist_G(u, v) = 2$ (contradiction).*

On peut conclure maintenant que $dist_G(u, v) \leq 2$, donc $diam_G\Omega \leq 2$ et de ce fait on a $length(T) \leq 2$ d'où $tl(G) \leq 2$.

□

3.4.2 Triangulations minimales et longueur arborescente

Dans cette partie, nous montrons la relation entre la longueur arborescente des graphes et les triangulations minimales.

Théorème 75 *[22]*
Soit G un graphe et Γ l'ensemble de toutes les triangulations minimales de G. Pour toute triangulation minimale $H \in \Gamma$ de G soit T_H l'arbre de cliques correspondant. Alors, il existe au moins une triangulation $H \in \Gamma$ telle que $tl(H) = tl(G)$ où $tl(H) = min_{T_H} length(T_H)$ et $length(T_H) = max_{X \in V(T_H)} diam_G(X)$.

Preuve
Soit G un graphe et $H \in \Gamma$ une triangulation minimale de G. Soit T_H l'arbre de cliques correspondant. Il est clair que l'on peut obtenir toutes les décompositions arborescentes de G en supprimant les arêtes ajoutées à l'arbre de cliques dans le processus de triangulation (il suffit de considérer la décomposition arborescente T du graphe G). Donc, par la définition 1.9.1*, les ensembles $T_X = \{i \in I, x \in X_i\}$ forment des sous arbres de T. Par ailleurs, considérons le graphe $H = (V, F)$ où $xy \in F$ si et seulement si x et y sont dans le même nœud X_i. Clairement, la famille $\{T_x, x \in X\}$ des sous arbres de T est un modèle d'intersection pour H et on obtient que H est un graphe triangulé. De plus, de la deuxième condition de la définition* 1.9.1*, toute arête de G est une arête dans H. Ainsi, H est une triangulation pour G. Finalement, il existe au moins une triangulation H telle que $tl(H) = tl(G)$, où $tl(H) = min_{T_H}\ length(T_H)$ et $length(T_H) = max_{X \in V(T_H)} diam_G(X)$ (i.e, le diamètre dans T_H est calculé dans G.*

□

Remarque 76 *On peut restreindre l'étude de la longueur arborescente parmi les triangulations minimales. Ainsi, le calcul de la longueur arborescente de tout graphe est équivalent à la recherche de la triangulation minimale H qui minimise le diamètre des nœuds de T.*

Dans cette partie, nous faisons une étude approfondie de l'algorithme $LB-Triang$ développé par Berry et al [10] avec l'objectif de l'adapter pour le calcul de la longueur arborescente des graphes.

Définition 77 *Un sommet x est dit $LB-$simplicial si et seulement si tout séparateur minimal contenu dans le voisinage de x est une clique.*

Définition 78 *La déficience d'un sommet x dans un graphe G, notée $Def_G(x)$, est un ensemble d'arêtes qui doit être ajouté à G pour rendre x simplicial. On définit la $LB-$ déficience d'un sommet x dans G, notée $LBDef_G(x)$, par l'ensemble des arêtes qui doit être ajouté à G pour rendre x $LB-$simplicial.*

Proposition 3.4.2 *[102]*
Un graphe G est triangulé si et seulement si tout sommet dans G est $LB-$ simplicial.

Proposition 3.4.3 *[98]*
Soit S et T deux séparateurs minimaux de G. Alors S croise T s'il existe deux composantes connexes $C_1, C_2 \in C(T)$, $C_1 \neq C_2$, telles que ; $S \cap C_1 \neq \emptyset$ et $S \cap C_2 \neq \emptyset$.

Proposition 3.4.4 *[10]*

Soit G un graphe et soit G' le graphe obtenu de G en saturant un ensemble S de séparateurs minimaux deux à deux non -croisés de G ; alors on a :

1. *Un séparateur clique minimal de G ne croise aucun séparateur minimal de G.*
2. *$\Im$ est un ensemble de séparateurs cliques minimales de G'.*
3. *Tout séparateur clique minimal de G est un séparateur minimal de G'.*
4. *Tout séparateur clique minimal de G' est un séparateur minimal de G.*
5. *Tout ensemble S de séparateurs minimaux deux à deux non -croisés de G' est un ensemble de séparateurs minimaux deux à deux non -croisés de G.*
6. *Si $\Im$ ensemble de séparateurs minimaux deux à deux non -croisés maximal de G alors G' est une triangulation minimale de G.*

L'algorithme LB-Triang [10] calcule une triangulation minimale en forçant chaque sommet à être $LB - simplicial$ par un ajout local d'arêtes.

Algorithm 1 Algorithme LB-Triang

Données: Un graphe $G = (V, E)$
Résultat: Une triangulation minimale H de G
pour *chaque* $x \in V$ **faire**
| Rendre x LB-simplicial
fin

Complexité

Soit G^{LB} le surgraphe obtenu après exécution de l'algorithme LB-Triang. La complexité de l'algorithme est en $O(nm')$, où m' est le nombre des arêtes de G^{LB}. À la fin d'une exécution, $\alpha = (x_1, x_2, \ldots, x_n)$ représentera l'ordre dans lequel les sommets ont été traîtés, et G^{LB} sera la triangulation obtenue. Notons que l'algorithme traite les sommets dans un ordre arbitraire.

3.4.3 Longueur arborescente des graphes k-Cordaux

Dans cette section, nous montrons que l'on peut trouver une tree décomposition de longueur au plus $\lceil \frac{k}{3} \rceil$ pour les graphes k - cordaux.

Algorithm 2 Algorithme LB-Triang modifié

Données: Un graphe $k-$ cordal $G = (V, E)$
Résultat: Une décomposition arborescente de longueur au plus $\lceil \frac{k}{3} \rceil$
pour $x \in V$ *tel que* $\forall u, v \in LBDef(x), dist_G(u, v) \leq \lceil \frac{k}{3} \rceil$ **faire**
| Rendre x LB-simplicial
fin

Soit l'exemple suivant :

La figure 3.3 représente un graphe k-cordal G avec sa triangulation minimale, où toute arête ajoutée (ici en rouge) est telle que ; $\forall (u, v) \in E^{'}, d_G(u, v) \leq \lceil k/3 \rceil$. Sur la droite de la figure, on a une décomposition arborescente de G qui correspond à l'arbre de cliques de la triangulation précédente. Cette décomposition arborescente est telle que $Length(T) = 2$. Ainsi, $tl(G) \leq \lceil k/3 \rceil = \lceil 6/3 \rceil = 2$.

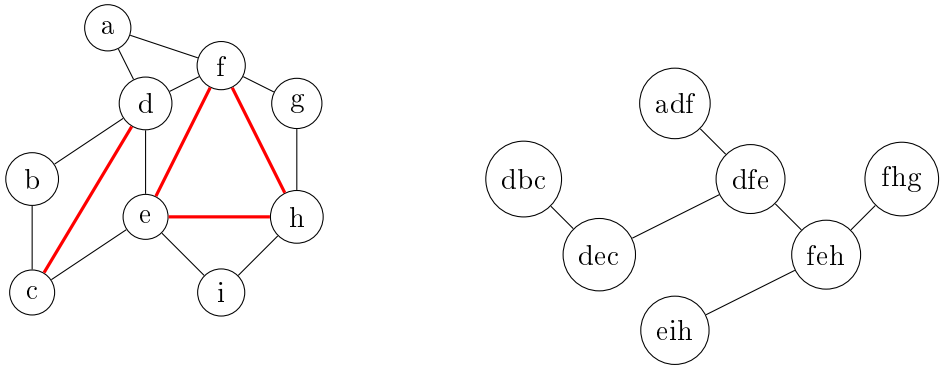


FIGURE 3.3 – Un graphe 6-cordal et sa décomposition arborescente

Dans ce qui suit, nous allons prouver que l'algorithme $LB - Triang$ modifié calcule une triangulation mninimale pour le graphe en entrée, et la décomposition arborescente est obtenue à partir de l'arbre de clique de la triangulation minimale, par une suppression des arêtes ajoutées.

Tout d'abord, nous montrons que l'on obtient une triangulation minimale.

Pour prouver ce théorème, nous aurons besoin de quelques lemmes techniques. Notons par (P) la propriété : $x \in V$ tel que $\forall u, v \in LBDef(x), dist_G(u, v) \leq \lceil \frac{k}{3} \rceil$.

Théorème 79 *L'algorithme calcule une triangulation minimale du graphe en entrée. De plus, tout sommet vérifie la propriété* (P).

Lemme 3.4.1 *[10]*

Soit G un graphe, et soit x un sommet de G. les séparateurs minimaux inclus dans $N(x)$ sont deux à deux non croisés dans G.

Lemme 3.4.2 *[10]*

Soit G un graphe, et G' le graphe obtenu de G en saturant un ensemble de séparateurs minimaux deux à deux non croisés de G, et soit x un sommet $LB-$ simplicial de G. Alors x est $LB-$ simplicial dans G'.

Lemme 3.4.3 *Soit G un graphe, et soit G' le graphe obtenu à partir de G en saturant un ensemble de séparateurs minimaux deux à deux non croisés 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 3.4.3). x vérifie la propriété (P) dans $G \Rightarrow x$ vérifie la propriété (P) dans G' . Evidemment $u, v \in LBDef_G(x)$, on a $dist_G(u, v) \leq \lceil \frac{k}{3} \rceil$. 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 cliques dans G'). Mais, il reste que $u, v \in LBDef_{G'}(x)$, on a $dist_G(u, v) \leq \lceil \frac{k}{3} \rceil$.

□

Lemme 3.4.4 *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 les étapes suivantes.*

Preuve

Pour $i = 1$ à n, par le lemme 3.4.2, G_{i+1} est obtenu à partir de G_i en saturant un ensemble de séparateurs minimaux deux à deux non croisés de G_i, par le lemme 3.4.4, 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 3.4.5 *Le graphe G^{LB} résultant de l'algorithme $LB-$ Triang est une triangulation de G.*

Preuve

Par le lemme 3.4.4, à la fin de l'exécution, tout sommet de G^{LB} est $LB-$simplicial. D'où, G^{LB} est triangulé.

□

Lemme 3.4.6 *[10]*

Soit G un graphe et soit G' le graphe obtenu de G en saturant un ensemble de séparateurs minimaux deux à deux non croisés de G. Si G' est triangulé alors G' est une triangulation minimale de G.

Preuve (du Théorème 3.4.1)

Par le lemme 3.4.6, on obtient un graphe qui est triangulé, et par le lemme 3.4.5, G^{LB} est obtenu de G en saturant un ensemble de séparateurs minimaux deux à deux non croisés de G. Ainsi G^{LB} est une triangulation minimale de G. De plus, tout sommet x de G vérifie la propriété (P).

□

Théorème 80 *La décomposition arborescente obtenue de l'arbre de cliques de la triangulation minimale, en supprimant les arêtes ajoutées est de longueur au plus $\lceil \frac{k}{3} \rceil$.*

Preuve

Soit $H = G^{LB}$ la triangulation minimale obtenue par l'algorithme et notons par T_H l'arbre de cliques de H. Supposons que $tl(H) > \lceil \frac{k}{3} \rceil$ alors on a :$Length(T_H) > \lceil \frac{k}{3} \rceil$. Ainsi, $Length(T_H) = \max_{B \in V(T_H)} diam_G(B) > \lceil \frac{k}{3} \rceil$. Et donc, il existe B tel que ; $\max_{u,v \in B} dist_G(u,v) > \lceil \frac{k}{3} \rceil$. Ainsi $\exists u, v \in LBDef(x_i), dist_G(u,v) > \lceil \frac{k}{3} \rceil$. Ainsi, il existe $u, v \in LBDef(x_i), dist_G(u,v) > \lceil \frac{k}{3} \rceil$. Et donc, il existe un cycle induit dans G tel que $tl(C) > \lceil k/3 \rceil$ (Contradiction).

□

Corollaire 3.4.1 *Tout graphe k-cordal est de longueur arborescente au plus $\lceil k/3 \rceil$.*

Chapitre 4

Algorithmes Dynamiques pour le Maintien de la Topologie des Graphes

Sommaire

Dans ce quatrème chapitre, nous nous intéressons aux concepts de l'algorithmiques dynamiques, qui incarnent la spécificité de cette thèse : le maintien dynamique de représentations de graphes lors de changements élémentaires du graphe en question. Nous introduisons d'abord la problématique de l'algorithmique dynamique et les notions qui s'y rapportent et ensuite nous présentons nos contributions dans ce domaine.

4.1 Problématique de l'algorithmique dynamique

Tout d'abord, il convient de noter que dans la thèse de Crespelle [44], il y est présenté au chapitre 2 un état de l'art très instructif sur la problématique de l'algorithmique dynamique des graphes. Dans cette section, nous proposons d'en faire un résumé non exhaustif fort utile à la bonne compréhension des différents outils utilisés dans la suite de notre thèse.

4.1.1 Les algorithmes dynamiques

L'algorithmique dynamique des graphes est une branche de la théorie des graphes dont l'objectif consiste à maintenir le résultat d'un calcul lorsque le graphe en question subit une modification. En fait, la différence entre un algorithme dynamique et un algorithme statique réside dans le fait que l'algorithme statique doit fournir le résultat du calcul à partir de la donnée du problème, mais ne possède pas pour ce faire le résultat du calcul sur un autre graphe.

Tout problème de graphe peut être considéré du point de vue dynamique. On peut par exemple mettre à jour, lorsque le graphe considéré est modifié, la liste des composantes connexes d'un graphe, un circuit hamiltonien, l'ordre de rencontre des sommets dans un parcours en profondeur, le résultat d'un problème de décision, etc.

La question fondamentale de l'algorithmique dynamique est donc la suivante : Connaissant le résultat d'un calcul sur un graphe et étant donné une modification de ce graphe, à quel coût peut - on déduire le résultat du calcul sur le nouveau graphe ?

On ne doit considérer que des modifications "légères" sur le graphe, car le graphe modifié sera ainsi assez proche du graphe initial. Dans ce cas, il sera possible de tirer parti du résultat afin de déduire le résultat du calcul pour le graphe modifié, à un coût moindre de celui du calcul statique sur le graphe modifié. Dans le cas d'une modification quelconque, on risquerait d'obtenir un graphe modifié assez différent du graphe initial, et qui aura pour conséquence, que notre calcul ne sera pas très différent d'un calcul statique et ainsi toute notre démarche n'aura plus de sens.

Dans la plupart des algorithmes dynamiques (dont ceux présentés dans cette thèse), les modifications élémentaires du graphe autorisées sont parmi les quatre suivantes :

- Ajout d'une arête entre deux sommets du graphe,
- Suppression d'une arête entre deux sommets du graphe,
- Ajout d'un nouveau sommet avec les arêtes définissant son voisinage,
- Suppression d'un sommet avec les arêtes qui lui sont incidentes.

Les algorithmes supportant les opérations d'ajout et de suppression sont dits entièrement dynamiques (En anglais : fully dynamic). Par exemple, on parle d'algorithme entièrement dynamique sur les sommets ou sur les arêtes.

4.1.2 Motivation

L'utilité des algorithmes dynamiques réside dans le fait que les graphes peuvent modéliser des systèmes qui se modifient dans le temps. On peut rencontrer ces modèles dans tous les domaines d'application des graphes mais prend une importance particulière dans le domaine des réseaux de communication.

Lorsque l'algorithmique dynamique est appliquée aux problèmes de réseaux, elle est généralement qualifiée d'algorithmique online.

Le lecteur désireux d'en savoir plus sur cette branche de la théorie se référera à [18, 57]. Le graphe physique d'internet et le graphe des communications directes dans un réseau adhoc sont des exemples de graphes ayant une forte dynamique. Dans ces cas où le graphe considéré se modifie très vite, la complexité du calcul dynamique est en compétition directe avec la fréquence moyenne des modifications du graphe. En effet, si on ne veut pas être submergé par les modifications, il faut être capable de traiter chacune d'entre elles suffisamment rapidement pour maintenir leur file d'arrivée à un niveau stable. Ainsi, les progrés de l'algorithmique dynamique pourraient fournir les outils nécessaires à l'observation de systèmes modélisables par des graphes et se modifiant à une fréquence élevée. Un autre domaine privilégié d'application de

l'algorithmique dynamique est celui de la gestion des structures de données employées dans les programmes informatiques. En effet, au cours de l'exécution d'un programme, il est rare que les structures de données soient figées, et il peut être nécessaire au cours de leur modification de savoir si elles vérifient une propriété donnée ou d'en entretenir une représentation particulière. Dès lors, les algorithmes utilisés sont dynamiques.

D'un point de vue théorique, l'algorithmique dynamique de graphe cherche à déterminer la stabilité d'un calcul à une modification de sa donnée. C'est - à - dire si une légère modification est opérée sur la donnée, le nouveau résultat recherché est il séparé du premier par un temps de calcul faible ou important ? Des problèmes de complexité très différentes ont été étudiés d'un point de vue dynamique : des problèmes NP-complets jusqu'aux problèmes pour lesquels on connait un algorithme de complexité linéaire. Néanmoins, même si le nombre d'algorithmes dynamiques est important, seule un petit nombre de problèmes ont concentré une attention particulière. C'est le cas des problèmes de connexité, de biconnexité, de triconnexité, de 2-arête connexité, ou encore les problèmes de bipartition, de fermeture transitive, de plus courts chemins et d'arbre couvrant de poids minimal.

L'intérêt du domaine est accentué par le fait que les problèmes de l'algorithmique des graphes ont des comportements dynamiques très variés. Certains sont naturellement dynamiques alors que pour d'autres, il paraît difficile de faire mieux que refaire entièrement le calcul sur le graphe modifié. Par exemple, parmi les problèmes difficiles à traiter dynamiquement de manière efficace, on trouve des problèmes de graphes extrêmement simples. C'est le cas du calcul des composantes connexes d'un graphe non orienté. Il existe un algorithme qui résout le problème en temps linéaire : il suffit de faire un parcours en largeur des sommets du graphe. Par contre, d'un point de vue dynamique, comment déterminer si une composante connexe est scindée en deux par le retrait d'une arête ? Faire un parcours en largeur aboutirait à une complexité dans le pire des cas qui est celle du problème statique. Pourtant, Holm, de Lichtenberg et Thorup [85] donnent un algorithme dynamique traitant l'ajout ou le retrait d'une arête en temps amorti $O(\log^2 n)$ par modification. Leur algorithme fait appel à des structures de données très complexes, et son analyse est difficile. La conception d'algorithmes dynamiques est souvent difficile et pousse à explorer plus loin les possibilités algorithmiques.

Définition des problèmes dynamiques de graphe

Il n'est pas si facile de définir un problème dynamique. Un problème statique, est défini par une donnée, qui sert de base au calcul, et une question, qui exprime ce qu'on attend du calcul. Un problème dynamique est défini comme un problème statique sauf que la donnée d'un problème dynamique comporte une difficulté supplémentaire. On distingue deux parties dans la donnée : la modification du graphe d'une part et la donnée résiduelle d'autre part.

La modification du graphe.

C'est une donnée objective du problème, le formalisme dans lequel elle est décrite fait partie intégrante de la définition du problème et n'est pas laissé au choix de celui qui le résoud. Comme dans le cas statique, la façon dont est codée la donnée, ici la modification du graphe, a une influence sur la complexité que peut atteindre un algorithme résolvant le problème. Par exemple, lors de l'ajout d'un sommet, si son voisinage est un ensemble de sommets consécutifs dans un ordre fixé, on peut ne donner que son premier et dernier voisin dans cet ordre. Ou encore, lors du retrait d'un sommet, on peut préciser ou non son voisinage. Dans toute la suite de cette thèse, nous n'exploitons pas la piste laissée par la question de la représentation de la modification du graphe : les arêtes sont données par leurs deux sommets incidents, et les voisinages, lors de l'ajout

d'un sommet, sont donnés par la liste explicite des sommets qu'ils contiennent. Lors du retrait d'un sommet, son voisinage n'est pas précisé. Reste une question en suspens concernant la façon dont est donnée la modification : un sommet est-il désigné par son identifiant ou par un pointeur sur une zone mémoire qui lui est propre ? Dans la littérature, la réponse à cette question est rarement précisée par les auteurs. Pourtant, elle n'est pas sans enjeux. Si le sommet est donné par son identifiant, il est souvent nécessaire de le rechercher dans la structure de donnée, alors que s'il est désigné par un pointeur sur la zone de la structure de donnée lui étant dédiée, cette recherche peut être évitée.

La donnée résiduelle.

C'est ce qui est présent en mémoire avant la modification demandée. Elle dépend entièrement de l'algorithme conçu pour répondre au problème et ne fait pas partie de la définition du problème. L'algorithme a le choix des structures qu'il entretient. La représentation du graphe adoptée peut être complète ou partielle, elle peut comporter tout objet jugé utile par le concepteur de l'algorithme. Le problème n'exprime aucune contrainte sur la structure employée, sa seule exigence est que l'algorithme réponde à la question posée pour le graphe résultant de la modification.

4.1.3 Le problème de représentation et reconnaissance dynamique d'une classe de graphes

Les problèmes de représentation et reconnaissance dynamique d'une classe de graphes (PRRD en abrégé) sont une sous-famille des problèmes dynamiques de graphes. Dans ce type de problème sur une classe de graphe F, la question posée est : le graphe modifié appartient il à F ? et si oui, en donner une représentation (fixée ou libre). Dans un PRRD, la représentation demandée requiert souvent la condition que le graphe appartienne à F. Par exemple, le problème de reconnaissance et de représentation dynamique pour la classe des graphes de permutation peut s'exprimer de la manière suivante : le graphe modifié est-il un graphe de permutation ? si oui, en donner un réaliseur. Dans ce cas, l'existence d'un réaliseur est, par définition, subordonnée à l'appartenance à la classe. Les PRRD présentent une restriction importante par rapport aux problèmes dynamiques en général. N'importe quelle séquence de modifications ne peut pas être envisagée, car l'algorithme s'arrête dès lors que le graphe n'appartient plus à F. Cette restriction qui peut paraître fortement contraignante est en fait difficile à éviter. En effet, l'hypothèse que le graphe avant la modification appartient à la classe est pour beaucoup dans la possibilité de déterminer efficacement si le graphe modifié est dans la classe. Ceci vient du fait que l'appartenance à la classe offre une représentation particulière du graphe et qu'ainsi le problème de savoir si le nouveau graphe est dans la classe se ramène à vérifier si la modification respecte la représentation propre à la classe. Si on s'autorise, lors de la série de modifications du graphe, à sortir de la classe, alors on peut être amené à considérer n'importe quel graphe et le problème prend une tout autre forme. Il s'agit maintenant de trouver une structure qui décrit non plus que le graphe a telle propriété mais plutôt qui mesure la "distance" à laquelle il est de cette propriété. L'exercice est beaucoup plus difficile, d'autant que l'objectif est toujours de faire mieux que la reconnaissance statique.

Le PRRD, dans sa définition présentée ici, a été étudié pour de nombreuses classes de graphes [120, 42, 108, 43, 82, 88, 86, 87, 45, 55, 90, 80, 91, 109, 106].

Pour en finir avec les notions de l'algorithmique dynamique, il reste à donner quelques repères sur les questions liées à la complexité des différents algorithmes dynamique.

4.2 Remarques sur la complexité des algorithmes dynamiques

Afin de donner des repères sur les complexités des différents algorithmes dynamiques dont il sera question dans cette thèse, on apporte quelques éléments de réponses aux questions suivantes : y a-t-il des opérations plus délicates que d'autres à traiter parmi les quatres modifications élémentaires ? est-il plus difficile de concevoir un algorithme entièrement dynamique qu'un algorithme monotone, (ne manipulant que les insertions ou que les suppressions) ? L'étude de la classe des graphes d'intervalles unitaires illustre à merveille certaines propriétés et comportements des algorithme dynamiques en rapport avec ces questions. Il existe un algorithme de reconnaissance et de représentation entièrement dynamique pour la classe des graphes d'intervalles unitaires [82]. Cet algorithme traite l'insertion d'un sommet de degrés d en $O(d + \log n)$, la suppression d'un sommet en $O(\log n)$, l'insertion d'arête en $O(\log n)$ et enfin la suppression d'arête en $O(1)$.

Remarquons d'abord la dissymétrie entre les cas d'insertion et de suppression. Ici, les insertions coûtent plus cher, mais dans d'autres problèmes dynamiques, le contraire peut arriver. Il n'y a en fait aucune raison pour que les complexités d'ajouts et de suppression soient les mêmes, ni pour qu'elles soient en faveur de l'une ou l'autre des opérations : cela dépend entièrement du problème traité. Plus intéressant encore, [82] propose un algorithme purement incrémental sur les arêtes où chaque insertion est traitée en temps $O(1)$ dans le pire des cas. Cela illustre le fait qu'un algorithme monotone, qui ne considère que des insertions ou que des suppressions, peut prétendre à une meilleure complexité dans le pire des cas qu'un algorithme qui manipule les deux opérations alternativement. Ceci dit, cela pourrait venir du fait qu'il est plus difficile de concevoir un algorithme entièrement dynamique qui atteint la même complexité que l'algorithme monotone, sans pour autant signifier qu'un tel algorithme dynamique n'existe pas. Dans le cas présent, [82] montre une borne inférieure de $(\log n/(\log\log n + \log b))$ par opération pour la reconnaissance entièrement dynamique sur les arêtes des graphes d'intervalles unitaires, dans le modèle de calcul "cell probe" avec une taille de mots b. Pour plus de détails sur le modèle de calcul et la technique de preuve utilisée, [82] renvoie à [131, 59, 83]. Ce résultat fournit l'exemple d'un problème dynamique dont la complexité d'un algorithme monotone, résolvant le problème soit pour l'insertion soit pour la suppression, ne peut pas être atteinte par un algorithme traitant à la fois l'insertion et la suppression.

4.3 Des listes d'adjacence dynamiques

Examinons le comportement des listes et de la matrice d'adjacence soumises à des modifications élémentaires du graphe. Les comportements dynamiques des deux structures sont sans rapport. Cela vient du fait que la matrice d'adjacence est un tableau alors que les lites d'adjacences sont composées principalement de listes, même si un tableau peut s'avérer utile pour leur implémentation.

Comportements dynamiques des listes et des tableaux

Les listes. Les listes n'offrent aucune résistance à l'insertion et à la suppression d'éléments, ce qui s'avère souvent avantageux pour les algorithmes dynamiques. Leur atout majeur est de permettre l'ajout et le retrait d'un élément à une position donnée en temps constant. Dit ainsi, les listes semblent la structure de donnée naturellement appropriée pour les algorithmes dynamiques. La situation est en réalité loin d'être si tranchée. La gestion dynamique des listes pose une difficulté qui provient du fait que l'on connait rarement la position à laquelle on veut effectuer

la modification. C'est pourquoi, dans beaucoup de situations, il est nécessaire de parcourir la liste afin de trouver cette position de modification, ce qui demande un temps proportionnel à la longueur de la liste.

Les tableaux. A l'opposé, le tableau est une structure purement statique très réticente aux modifications mais qui permet une recherche rapide. L'avantage et l'essence même du tableau est de permettre un accès en temps constant au contenu d'une de ses cases dont on connait l'index. Cette propriété remarquable est rendue possible par le fait que la partie de la mémoire occupée par le tableau est formée de cases mémoires consécutives. Il n'est donc pas possible d'insérer ou de supprimer une case au milieu d'un tableau sans déplacer toutes les cases précédant ou suivant la position d'insertion ou de suppression. Pire, il n'est pas non plus possible d'ajouter une case en début ou en fin du tableau de manière efficace car rien ne garantit que les cases mémoires qui jouxtent la zone occupée par le tableau soient libres. La seule possibilité consiste alors à déplacer entièrement le tableau vers une zone mémoire offrant le nombre suffisant de cases mémoires libres et consécutives. Ainsi, les opérations d'ajout et de suppression d'une case dans un tableau prennent toutes deux un temps $O(l)$ dans le pire des cas, où l est le nombre de cases du tableau, car le déplacement entier du tableau peut être nécessaire. Pour les tableaux à plusieurs dimensions, la situation n'est pas différente : le retrait ou l'ajout d'une "ligne" demande de réécrire entièrement le tableau dans le pire des cas.

Entretien dynamique de la matrice d'adjacence

En ce qui concerne la matrice d'adjacence qui est un tableau à 2 dimensions et n^2 cases, le retrait ou l'ajout d'un sommet implique d'enlever ou d'ajouter une ligne et une colonne à la matrice, et donc de déplacer toute la structure. Le temps demandé pour mettre à jour la matrice d'adjacence lors d'une modification de sommet est $\Omega(n^2)$. Inversement, du fait de l'accès en temps constant à une case, les modifications d'arêtes ne demandent qu'un temps $O(1)$. Pour garantir cet accès en temps constant à partir de la donnée des identifiants des sommets, il est nécessaire de ré-attribuer ces identifiants, après chaque retrait de sommet, de sorte que tous les numéros utilisés restent consécutifs.

Entretien dynamique des listes d'adjacence

Les listes d'adjacence peuvent être implémentées de plusieurs manières. La liste des sommets du graphe peut être stockée dans un tableau, dans une liste, dans un arbre de recherche ou toute autre structure ; de même pour la liste des voisins d'un sommet. Quelle que soit l'implémentation choisie, son comportement dynamique peut être amélioré en introduisant quelques pointeurs supplémentaires que nous appelerons des pointeurs de position. Lorsque un sommet x est présent dans la liste des voisins d'un sommet y, on ajoute un pointeur de la cellule de x dans la liste de y vers la cellule de y dans la liste de x. Ainsi, lors de la suppression de x, on peut accéder en temps constant à sa cellule dans la liste de y et l'en supprimer. La seule des implémentations des listes d'adjacence présentées ici qui fournisse la fonctionnalité de donner la liste des voisins d'un sommet x en un temps proportionnel à son degrés est celle qui utilise un tableau pour stocker la liste des sommets du graphe. Dans les autres implémentations, le coût d'une telle requête est augmenté de celui de la recherche de x dans la liste des sommets du graphe. Cependant, la difficulté d'entretien dynamique du tableau incite à envisager d'autres implémentations plus faciles à actualiser.

L'implémentation classique. Commençons par l'implémentation la plus classique des listes d'adjacence : celle utilisant un tableau. Sous les modifications d'arête, cette structure est relativement aisée à maintenir. Lorsque la suppression de l'arête xy est demandée, il faut supprimer x dans la liste des voisins de y et vice versa. Cela peut être fait en temps $O(\min(d(x), d(y)))$ en parcourant les deux listes simultanément jusqu'à trouver y dans la liste des voisins de x, ou x dans la liste des voisins de y. Les pointeurs de position entre x et y permettent alors de trouver l'autre cellule recherchée. L'ajout d'une arête peut se faire en temps constant (si on suppose que l'arête n'existe pas déjà) car on peut ajouter le nouveau voisin de x et celui de y au début de leurs listes respectives, avec les pointeurs de position nécessaires. En ce qui concerne les modifications de sommet, comme déjà évoqué, la difficulté d'entretien vient de la nécessité d'ajouter ou de retirer une case du tableau, ce qui nécessite un temps $O(n)$. Le reste de l'actualisation peut être fait plus rapidement. Ajouter le sommet x et la liste de ses voisins prends un temps $O(d(x))$; et ajouter x dans les listes respectives de ses voisins, avec les pointeurs de position correspondants, peut également se faire en $O(d(x))$ en ajoutant x au début de chaque liste. Le temps total demandé par un ajout de sommet est donc $O(n)$. Lors du retrait, effacer x des listes de ses voisins peut être fait en temps $O(d(x))$ grâce aux pointeurs de position qui donne accès en temps constant à la position de x dans la liste d'un quelconque de ses voisins. Le temps total de l'opération de retrait est donc $O(n)$, car il faut toujours modifier le tableau stockant le voisinage de chaque sommet.

4.4 Maintien de la topologie des graphes

Nous considérons le problème du maintien de quelques paramètres de graphes qui interviennent dans les décompositions arborescentes, en l'occurence la largeur et la longueur arborescente. Plus précisément, nous proposons un algorithme dynamique qui prend en charge les opérations de suppression et d'ajout d'arêtes ou de sommets. Nous mettons à jour les changements sur ces paramètres pour la classe des graphes faiblement triangulés dès que ces opérations sont effectuées. Une implémentation de ces algorithmes est présentée. De plus, on généralise cet algorithme pour les graphes quelconques.

On a vu dans les chapitres précédents que la décomposition arborescente des graphes est un concept fondamental en théorie des graphes au même titre que la largeur arborescente et la longueur arborescente.

La classe des graphes triangulés est une classes de graphes très étudiée. L'une des raisons en est que cette classe admet une décomposition arborescente naturelle appelée arbre de cliques (en anglais ; clique tree), qui peut être construit en temps linéaire [50, 65, 68]. Le calcul de la longueur et de la largeur arborescente pour cette classe est connu (au plus $\omega(G)$ pour la largeur arborescente, où $\omega(G)$ est la taille de la plus large clique, et la longueur arborescente est égal à un).

La classe des graphes faiblement triangulés est aussi une classe bien étudiée et a de nombreuses applications connues [35, 77, 123]. Cette classe de graphes a été introduite par Hayward [75], et est une sous classe des graphes parfaits qui inclut les graphes triangulés et leurs complémentaires.

Le premier algorithme dynamique qui maintient la représentation arbre de cliques des graphes triangulés a été introduit par L. Ibarra [87]. Cet algorithme supporte les opérations de suppression et ajoût d'arêtes et de sommets. Nous avons généralisé dans [29] ce résultat, en présentant un algorithme qui maintien quelques propriétés sur les faiblements triangulés.

Dans cette section, nous présentons un algorithme entièrement dynamique pour les graphes faiblement triangulés et qui supporte les opérations de suppression et d'ajoût d'arêtes et de

sommets. De plus, nous mettons à jour les paramètres largeur et longueur arborescente quand des changements dans la topologie surviennent.

Nous utiliserons les conventions suivantes : Nous nous réfererons aux sommets pour G et aux nœuds pour T et on utilisera s, t, u, v comme noms de sommets et W, X, Y, Z comme noms pour les nœuds respectivement. Un nœud $X \in T$ correspond à un sous graphe induit B_X de G. On note par $G + (x, y)$ (resp. $G - (x, y)$), le graphe obtenu en ajoutant (resp. supprimant) une arête (x, y). On note par $G + x$ (resp. $G - x$), le graphe obtenu en ajoutant (resp. supprimant) un sommet x.

Tous les graphes considérés dans la suite sont connexes. Pour maintenir la représentation décomposition arborescente d'un graphe non connexe dans notre algorithme dynamique, on étend la définition comme suit. Soit G un graphe non connexe et soit $W_G = T_1, T_2, ..., T_k$ la forêt des décompositions arborescentes des k-composantes connexes de G, $(k > 1)$. Soit X et Y deux sacs de deux différents T_i, tels que B_X et B_Y sont contenues dans différentes composantes connexes de G. Puisque $B_X \cap B_Y = \emptyset$, et $\{X, Y\}$ n'est pas une arête de W_G, on appelle $\{X, Y\}$ une arête nulle de poids $w(X, Y) = 0$. On relie $T_1, T_2, ..., T_k$ avec les arêtes arbitraires nulles pour former un arbre T, qui satisfait la propriété des sous arbres induits de la définition 1.9.1. On étend W_G en lui ajoutant des arêtes nulles, telles que T devienne un arbre couvrant de poids maximum à partir de W_G. Ainsi, T satisfait aux trois propriétés de décomposition arborescente. On utilisera le terme de décomposition arborescente pour référer à T et le terme de décomposition arborescente stricte pour référer à T_i, qui sont des sous arbres de T correspondant aux composantes connexes de G.

4.4.1 Maintien des graphes triangulés

Dans toute la suite, une requête désignera une question en rapport avec le maintien d'une solution.

Ibarra [87] a donné le premier algorithme dynamique qui maintient la représentation arbre de cliques d'un graphe triangulé G et supporte les opérations suivantes :

- Requête-supprimer(u, v) retourne "oui" si $G - (u, v)$ est triangulé et "non" sinon.
- Requête-ajouter(u, v) retourne "oui" si $G + (u, v)$ est triangulé et "non" sinon.

L'Algorithme dynamique pour les graphes triangulés

Dans ce qui suit, G désigne le graphe triangulé courant, et T son arbre de clique.

A) Requête-supprimer(u, v)

On montre ici comment décider si $G - (u, v)$ est triangulé pour $(u, v) \in G$.

Lemme 4.4.1 *[118]*
Soit G un graphe triangulé avec l'arête (u, v). Alors on a soit $G - (u, v)$ est triangulé soit $G - (u, v)$ possède un cycle sans cordes de longueur 4.

Lemme 4.4.2 *[70]*
Un graphe H ne possède pas de cycles de longueur 4 si et seulement si pour tout sommets distincts s, t avec $(s, t) \notin E(H)$, $H + (s, t)$ possède exactement une clique maximale contenant (s, t).

Théorème 81 *[87]*
Soit G un graphe triangulé avec l'arête (u, v). Alors $G - (u, v)$ est triangulé si et seulement si G possède exactement une clique maximale contenant (u, v).

Procédure Requête-supprimer(u, v)
Pour tout nœud x de T, tester si $(u, v) \in B_X$. S'il existe exactement un tel nœud, retourner "oui", et sinon retourner "non".
Fin Requête-supprimer

B) Requête-ajouter(u, v)

Dans ce qui suit Ibarra (1999) montre comment décider si $G + (u, v)$ est triangulé pour $(u, v) \notin E$. Il utilise le fait que si T_u et T_v sont des sous arbres de T respectivement induits par $B(u)$ et $B(v)$ alors T_u et T_v ne s'intersectent pas, parce qu'aucune clique ne contient u et v.

Théorème 82 *[87]*
Soit G un graphe triangulé sans l'arête (u, v). Alors $G + (u, v)$ est triangulé si et seulement si, il existe un arbre de cliques T de G tel que $u \in B_X$ et $v \in B_Y$ pour $(X, Y) \in T$.

Théorème 83 *[87]*
Soit G un graphe triangulé sans l'arête (u, v). Soit T un arbre de cliques de G et soit X et Y les nœuds dans T tels que $u \in B_X$ et $v \in B_Y$. Supposons que $(X, Y) \in T$. Alors, Il existe un arbre de cliques T' de G tel que $u \in B_{X'}$ et $v \in B_{Y'}$ et $(X', Y') \in T'$ si et seulement si le poids minimum d'une arête e sur une X, Y- chaîne dans T satisfait $w(e) = w(X, Y)$.

Procédure Requête-insérer(u, v)
Trouver les nœuds X, Y de T, tels que $u \in B_X$ et $v \in B_Y$. **Si** $(X, Y) \in T$. Retourner "oui".
Sinon, trouver le poids minimum d'une arête e sur une $X - Y$ chaîne dans T. **Si** $w(e) > w(X, Y)$, Retourner "non".
Fin Requête-insérer (u, v)

Ibarra [87] a donné une implémentation de la procédure **Requête-insérer** (u, v) en temps $O(\log^2 n)$ et de **Requête-supprimer** (u, v) en temps $O(n)$.

4.4.2 Algorithme dynamique pour les graphes faiblement triangulés

Modifications sur les arêtes

A)-Insertion d'une arête

Dans cette partie, nous donnons quelques observations sur le cas d'une insertion d'une arête au graphe faiblement triangulé.

Lemme 4.4.3 *[123]*
Soit $G = (V, E)$ un graphe faiblement triangulé, une 2 - paire $\{x, y\}$, alors le graphe obtenu par l'insertion de l'arête (x, y) à G est faiblement triangulé.

Maintenant, supposons que les extrémités de (x, y) ne forment pas une 2 - paire.

Lemme 4.4.4 *[7]*
Étant donné un graphe faiblement triangulé $G = (V, E)$, alors le graphe obtenu en ajoutant une arête (x, y) à G est faiblement triangulé si et seulement si (x, y) n'est pas l'arête du milieu d'un P_4 dans G.

B)-Suppression d'une arête

Les observations suivantes sont certes évidentes mais sont importantes pour le maintien du graphe faiblement triangulé en cas de suppression d'une arête au graphe.

Observation 84 *Si $G - (x, y)$ contient un C_5 alors (x, y) est l'unique corde d'un C_5 dans G.*

Observation 85 *Si $G - (x, y)$ contient un C_6 alors (x, y) est l'unique corde d'un C_6 dans G et appartient à au moins deux C_4.*

Observation 86 *Il est impossible que $G - (x, y)$ contienne un C_k $k \geq 7$.*

Preuve
Si $G - (x, y)$ contient un cycle C_k, $k \geq 7$, alors (x, y) est l'unique corde d'un C_7 dans G qui contient un C_5 (contradiction avec le fait que G est faiblement triangulé.

□

Proposition 4.4.1 *Étant donné un graphe triangulé $G = (V, E)$, alors le graphe obtenu par la suppression d'une arête (x, y) à G n'est pas faiblement triangulé si et seulement si (x, y) est l'unique corde d'un C_5 ou d'un C_6 dans G.*

Preuve
La condition nécessaire vient des Observations précédentes. Pour prouver la condition suffisante, soit $e = (x, y)$ une arête de G qui est l'unique corde d'un C_5 (respectivement d'un C_6). Alors par l'observation 4.4.5, toute arête de C_5 (resp.C_6) ne peut être LB-simplicial et par l'observation 4.4.6, $G - (x, y)$ n'est pas faiblement triangulé.

□

Modifications sur les sommets

C)-Insertion d'un sommet

Soit $G = (V, E)$ un graphe faiblement triangulé. Étant donné x un sommet incident aux sommets de l'ensemble $N(x) \subseteq V$, l'insertion de x à G induit un graphe $G + x = (V \cup \{x\}, E \cup \{xy / y \in N(x)\})$.

Nous donnons dans ce qui suit une caractérisation sur le maintien du graphe en cas d'insertion d'un sommet.

Observation 87 *Soit $E_x = \{xy / y \in N(x)\}$, alors $G + x$ est faiblement triangulé si et seulement si toute arête de E_x est LB-simplicial.*

Preuve
Par la proposition 4.4.1.

□

D)-Suppression d'un sommet

Soit $G = (V, E)$ un graphe faiblement triangulé, et un sommet x incident à un ensemble $N(x) \subseteq V$. La suppression de x de G induit un graphe $G - x = (V - \{x\}, E - \{xy/y \in N(x)\})$.
La caractérisation suivante est évidente.

Observation 88 *Soit $x \in V$, alors $G - x$ est faiblement triangulé.*

Preuve
Vient de l'hérédité dans les graphes faiblement triangulé.

□

4.4.3 Algorithme entièrement dynamique pour les graphes faiblements triangulés

Comme mentionné dans ce qui précède, notre algorithme maintient la représentation décomposition arborescente des graphes faiblement triangulés pour les différentes modifications Ajout et suppression d'une arête ou d'un sommet.

A)-Insertion d'une arête

Soit (u, v) l'arête à ajouter, et soit $G' = G + (u, v)$. Nous montrons comment mettre à jour T pour $G + (u, v)$. Soit T' la décomposition arborescente de G'. On a trois cas :

Cas 1. $\exists B_X$ tel que $(u, v) \in B_X$
B_X est de la forme $(N(s) \cap N(t)) \cup \{s\}$ ou $(N(s) \cap N(t)) \cup \{t\}$ avec $\{x, y\}$ une 2-pair. Alors, pas de changements $T = T'$, $tw(G) = tw(G')$ et $tl(G) = tl(G')$.

Cas 2. $u \in B_X$, $v \in B_Y$ et $(X, Y) \in E(T)$.

Proposition 4.4.2 *Soit $I = B_x \cap B_y$, alors il existe une décomposition arborescente T' de G' où $I \cup \{u, v\}$ est un sac de diamêtre au plus* 2.

Preuve
Soit T' un arbre défini par $T + (I \cup \{u, v\})$. Alors, on a $I \cup \{u, v\}$ qui ne contient que les sommets de G. L'arête $(u, v) \in G' = G + (u, v)$ implique que les conditions 1 *et* 2 *de la définition 1.9.1 sont satisfaites pour T' et la troisième condition de la définition 1.9.1 est satisfaite par définition de I. Alors, $I \cup \{u, v\}$ est un sac dans T'. De plus, u et v sont adjacents dans $I \cup \{u, v\}$ et $\forall s, t \in I$ on a $d_I(s, t) \leq 2$. Ainsi $\forall s, t \in I \cup \{u, v\}$, d'où $d_{I \cup \{u,v\}}(s, t) \leq 2$.*

□

Et nous avons les résultats suivants concernant la largeur et la longueur arborescente de $G' = G + (u, v)$.

Proposition 4.4.3 *On a $tw(G') \leq tw(G)$ et $tl(G') \leq 2$.*

Preuve

Puisque on a $Largeur(T) = max_{X \in V(T)} |X| - 1$, *si* B_z *est l'unique nœud qui détermine la largeur arborescente de* T, *alors* $tw(G') = tw(G)$. *De plus on a* $tl(G') \leq 2$ *qui découle de prop4.4.2 (i.e., la longueur arborescente n'augmente pas).*

□

Cas 3. $(X, Y) \notin T$.

Nous allons explicitement utiliser le fait que si T_u et T_v sont des sous arbres de T induits par $B_G(u)$ et $B_G(v)$ respectivement [7], alors T_u et T_v ne s'intersectent pas (puisque aucun sac ne contient (u, v)).

Soit $T_{\overline{uv}}$ un $B_G(u)$,$B_G(v)$-séparateur minimal, alors $T_{\overline{uv}}$ est un séparateur minimal dans T. De plus, $T_{\overline{uv}}$ est un sous arbre de T induit par les nœuds du séparateur minimal.

Proposition 4.4.4 *Pour* T' *la décomposition arborescente de* G' *avec* $(X, Y) \notin T$, *alors on a ;* $tw(G') \leq tw(G) + 1$ *et* $tl(G') \leq 2$.

Preuve

Si T_u *contient au moins un sac avec* $(tw(G) + 1)$ *sommets, alors l'insertion d'un sommet* v *aux sacs de* T_u *implique que* T_u *contiendra au moins un sac avec* $(tw(G) + 2)$ *sommets et ainsi* $tw(G') = tw(G) + 1$, *autrement,* $tw(G') = tw(G)$. *Même raisonnement pour* T_v. $tl(G') \leq tl(G)$ *; il est clair que l'insertion d'une arête peut diminuer les distances dans le graphe* G' *et par voie de conséquence la longueur de* T', *la borne sur la longueur arborescente de* G' *est la même que pour* G.

□

Algorithm 3 Algorithme Insérer Arête

Début **si** $\exists B_X$ *tel que* $uv \in B_X$ **alors**
| pas de changement $T = T'$, $tw(G) = tw(G')$ et $tl(G) = tl(G')$
finsi
si $(X, Y) \in T$ **alors**
| Trouver l'ensemble de nœuds $X, Y \in T$ tel que $u \in B_x, v \in B_y$ Remplacer l'arête (X, Y) dans T avec le nouveau nœud Z représentant $B_z = I \cup \{u, v\}$ et ajouter les arêtes $(X, Z) et (Z, Y)$
finsi
si $(X, Y) \in T$ **alors**
| Trouver les nœuds $X, Y \in T$ tels que $u \in B_x$, $v \in B_y$ trouver T_u et T_v les sous arbres de T respectivement induit par $B_G(u)$ et $B_G(v)$ et trouver $T_{\overline{uv}}$ le séparateur de X et Y dans T Comparer entre $|T_u|$ et $|T_v|$
| **si** $|T_u| \leq |T_v|$ **alors**
| | ajouter le sommet v à chaque sac de $T_{\overline{uv}}$ et B_x
| **finsi**
| **alors**
| | Ajouter le sommet u à chaque sac de $T_{\overline{uv}}$ et B_y
| **finsi**
finsi

7. $B_G(u)$: sacs qui contiennent u et $B_G(v)$: sacs qui contiennent v.

Algorithme Insérer Arête met à jour T, afin que T' soit une décomposition arborescente pour $G+\{u,v\}$. Par ailleurs, T' vérifie la propriété des sous arbres induits et les deux extrémités de l'arête $\{u,v\}$ appartiennent à $Z \in T'$, ce qui vérifie les trois conditions de la définition 1.9.1.

B)-Suppression d'une arête

Soit (u,v) l'arête à supprimer, et soit $G' = G-(u,v)$. Nous montrons comment mettre à jour T pour G'. Soit T' la décomposition arborescente de G'.

Algorithm 4 Algorithme Supprimer Arête

Début **si** *$uv \in B_X$ tel que B_X est une clique* **alors**
| Pas de changements, $T = T'$
finsi
si *$uv \in B_X$ tel que B_X est une non-clique* **alors**
| B_X est de la forme $(N(s) \cap N(t)) \cup \{s\}$ ou $(N(s) \cap N(t)) \cup \{t\}$ avec $\{s,t\}$ une 2-paire
| Partitionner l'ensemble $N(X)$ des X's voisins en $N_u = \{Y \in N(X) \ / \ u \in B_y\}$ $N_v = \{Z \in N(X) \ / v \in B_z\}$ $N_{\overline{uv}} = \{W \in N(X) / u,v \notin B_w\}$
finsi
pour *chaque nœud X de T tel que $(u,v) \in B_x$ soit $B_x^u = B_x - \{v\}$, $B_x^v = B_x - \{u\}$ Dans $G-(u,v)$, tout sac B_x est décomposé en deux sacs B_x^u et B_x^v* **faire**
| Remplacer le nœud X par deux nœuds X_1 and X_2, respectivement représentant B_x^u et B_x^v et ajouter l'arête (X_1, X_2)
fin
si $Y \in N_u$ **alors**
| Repmlacer (X,Y) par (X_1,Y)
finsi
si $Z \in N_v$ **alors**
| Repmlacer (X,Z) par (X_2,Z)
finsi
si $W \in N_{\overline{uv}}$ **alors**
| Remplacer (X,W) par (X_1,W) ou (X_2,W) choisis arbitrairement
finsi

On observe que T' vérifie la propriété des sous arbres induits. Ainsi, on a la proposition suivante :

Proposition 4.4.5 *$tw(G') \leq tw(G)$ et $tl(G') \leq 2$.*

Preuve

On a $width(T) = max_{X \in V(T)} |X| - 1$, si B_x est l'unique corde qui détermine la largeur arborescente de T, alors, le fait qu'on décompose B_x en deux sacs B_x^u et B_x^v, réduit la largeur de T, et on a $tw(G') < tw(G)$. Sinon $tw(G') = tw(G)$. Par ailleurs, on a $tl(G') \leq 2$; en effet, il est clair que la suppression d'une arête crée deux sacs B_x^u et B_x^v de telle manière que le diamètre de chaque sac ne dépasse pas 2.

□

C)-Insertion d'un sommet

Soit u le sommet à ajouter, et soit $G' = G+u = (V+u, E \cup \{\{u,v_i\}; v_i \in N_{G'}(u) \text{ et } v_i \in V(G)\}$. Nous montrons comment mettre à jour T pour G'. Soit T' la décomposition arborescente de G'.

Algorithme Insérer Sommet

Algorithm 5 Algorithme Insérer Sommet

Début **si** *il existe* $X \in T$ *tel que* $v_i \in B_x$, $\forall i$ **alors**
| $X' = B_x \cup \{u\}$ est un sac de T'
finsi
Insérer u à chaque nœud dans T
pour *tout nœud* X *qui est pendant dans* T **faire**
| Supprimer X tant qu'il existe au moins un nœud qui contient u et v_i, $\forall i$
fin

À la fin de la procédure, on obtient un sous arbre noté T_u induit par $B_{G'}(u)$. Ainsi, T' est obtenu à partir de T par la mise à jour des sacs dans les seuls sous arbres T_u , les autres sous arbres restent inchangés.

T' est une décomposition arborescente pour $G + u$ satisfaisant la propriété des sous arbres induits. Et on a le résultat suivant ;

Proposition 4.4.6 *On a ;* $tw(G) \leq tw(G') \leq tw(G) + 1$ *et* $tl(G') \leq 3$.

Preuve

Si B_x *est l'unique nœud qui détermine la largeur arborescente de* T*, alors le fait que l'on ajoute un nouveau sommet à* B_x *va impliquer que* $tw(G') = tw(G) + 1$*, sinon, on a,* $tw(G') = tw(G)$*. On a :* $tl(G') \leq Length(T')$*, et* $Length(T') = Max_{B \in V(T')} diam_{G'} B$ *et* $LengthT'_x = Max_{B_x \in V(T')} diam_{G'} B_x$ *avec* B_x *sac contenant* u *dans* T'*, et* $diam_{G'} B_x = max_{v,u \in B_x}\{dist_{G'}(v,u)\} \leq max_{v,u \in B_x}\{dist_G(v,u)\} + 1$ *qui implique que* $tl(G') \leq tl(G) + 1 \leq 3$.

□

D)-Suppression d'un sommet

Soit u le sommet à supprimer, et soit $G' = G - u$. Nous montrons comment mettre à jour T pour G'. Soit T' la décomposition arborescente de G'.

Pour chaque nœud $X \in T$ tel que $u \in B_x$, soit $B_{x'} = B_x - \{u\}$. Dans $G - \{u\}$, tout sac B_x est remplacé par $B_{x'}$.

Algorithme Supprimer Sommet

Algorithm 6 Algorithme Supprimer Sommet

Début **si** *il existe* B_{y_i} *tel que* $B_{x'} \subset B_{y_i}$*, pour quelques* $Y_i \in N(X')$ **alors**
| Choisir un tel Y_i arbitrairement Contracter $\{X', Y_i\}$ et Remplacer X' avec Y_i
finsi
pour *tout nœud* X *qui est pendant dans* T **faire**
| Supprimer X tant qu'il existe au moins un nœud qui contient u et v_i, $\forall i$
fin

La décomposition arborescente obtenue sera inévitablement réduite.

Proposition 4.4.7 *On a ;* $tw(G) - 1 \leq tw(G') \leq tw(G)$ *et* $tl(G') \leq 2$

Preuve

En effet, si tous les sacs de T qui contiennent $(tw(G)+1)$ sommets sont parmi les sacs auxquels on supprime le sommet u, alors ; $tw(G') = tw(G) - 1$. Sinon $tw(G') = tw(G)$. D'autre part, il est clair que, $tl(G') \leq tl(G) \leq 2$ (par définition).

□

4.4.4 Implémentation

Structure de donnée

Étant donné un graphe faiblement triangulé G en entrée, représenté par sa liste d'adjacence, comme une étape initiale de pré-traitement, on peut trouver les degrés de tous les sommets en temps linéaire et stocker les valeurs dans une liste. Puisque l'on suppose que la représentation décomposition arborescente du graphe est donnée, les sommets peuvent être partitionnés en sacs X_i selon la décomposition arborescente T en temps linéaire. Chaque élément dans la liste des degrés, indique son sac correspondant et le sommet dans le graphe.

De cette manière, trouver à quel ensemble appartient un sommet ou son degré correspondant, supprimer un sommet donné d'un ensemble, et ajouter un sommet donné à un ensemble se fait en temps constant. Par conséquence, le déplacement d'un ensemble de sommet donné A, se fait en temps $O(|A|)$.

Étant donné cette structure, scanner ou retourner un sommet dans la décomposition arborescente se fait en temps linéaire en la taille du nœud, puisque les listes des degrés sont toutes connectées par des pointeurs , donc on peut juste les scanner l'un après l'autre, en commençant par celui correspondant au degré minimum dans le vecteur.

Comme noté précédemment, la taille de la liste des degrés est n, mais puique nous permettons des ajouts de sommets au graphe, pour chaque opération nous devrions augmenter la taille de la liste par 1. Cette opération peut être amortie en temps constant en utilisant les listes dynamiques, ou l'allocation dès le début de la liste de taille $n+n'$, où n' est une borne supérieure au nombre des sommets ajoutés possibles donné par l'utilisateur de l'algorithme. Notons que ceci est quelque chose dont tout algorithme dynamique doit faire face indépendemment de la décomposition arborescente.

Operations

Dans cette section, nous montrons comment notre structure de donnée peut être mise à jour efficacement après une modification qui maintient la représentation décomposition arborescente du graphe en entrée en un temps raisonnable quand on ajoute ou supprime une arête ou un sommet.

Avant de commencer l'implémentation, nous avons besoin de connaître la complexité de la détermination des a, b-séparateurs minimaux dans les graphes faiblement triangulés qui est donné dans le papier de (Bouchitté et Todinca) [19]. Les auteurs ont donné un algorithme efficace qui liste tous les séparateurs minimaux des graphes faiblement triangulés. Une implémentation de cet algorithme est donnée dans [122]. L'algorithme détermine tous les a, b-séparateurs minimaux en temps $O(m^2)$, où m est le nombre des arêtes de G.

Rappelons cependant que l'on suppose que la représentation décomposition arborescente T du graphe G existe déjà.

Nous implémentons la procédure **Insérer Arête**, par la recherche des nœuds les plus proches X et Y tels que $u \in B_X$ et $v \in B_Y$, et déterminer si $(X, Y) \in T$ ou non. Ceci se fait en temps

polynomial. On a deux cas : Si $(X,Y) \notin T$, on calcule T_u et T_v les sous arbres de T, et le $X,Y-$séparateur minimal dans T , qui est fait en temps polynomial, et si $(X,Y) \notin T$, on remplace l'arête (X,Y) avec le nœud Z et on ajoute les arêtes (X,Z) et (Z,Y). Puisque T a au plus n nœuds et chaque vecteur est de longueur n, la procédure d'insertion d'une arête se fait en temps $O(m^2)$.

Nous implémentons la procédure **Supprimer Arête** en trouvant le nœud X et ensuite, on examine les X voisins en un temps polynomial.

Nous implémentons la procédure **Insérer Sommet** en trouvant les nœuds X, et ajoutant u à chacun d'entre eux, ensuite, on le supprime quand il existe au moins un nœud qui contient u et v_i , où $v_i \in E^{'}$. Ceci est fait en temps polynomial.

Nous implémentons la procédure **Supprimer Sommet** en trouvant les nœuds X, et on supprime u de ces sacs. Ceci est fait en temps polynomial.

Finallement, le coût total de notre algorithme dynamique est $O(m^2)$.

4.4.5 Maintien des graphes quelconques

Dans cette partie, nous présentons un algorithme qui maintient la représentation en décomposition arborescente pour les graphes en général et met à jour les changements sur les invariants, longueur et largeur arborescente du graphe, quand on supprimme ou on ajoute une arête ou un sommet. Notons que les résultats de cette partie ont été publiés dans [23, 26].

i)-Insertion d'une arête

Soit (u,v) l'arête à ajouter, et soit $G' = G + (u,v)$. Nous montrons comment mettre à jour T pour G'. Soit T' la décomposition arborescente de G'.

Soit $u \in B_X$, $v \in B_Y$ et $(X,Y) \in T$.

Proposition 4.4.8 *Soit $I = B_x \cap B_y$, alors il existe une décomposition arborescente T' de $G + (u,v)$ où $I \cup \{u,v\}$ est un sac.*

Preuve

Soit T' un arbre défini par $T \cup I \cup \{u,v\}$. Alors, on a que $I \cup \{u,v\}$ contient seulement les sommets de G. L'arête $(u,v) \in G + (u,v)$ implique que la condition 1 et 2 de la définition 1.9.1 sont satisfaites pour T' et la troisième condition de la définition est satisfaite par définition de I. Donc, $I \cup \{u,v\}$ est un sac dans T'.

□

Deux cas sont considérés : $(X,Y) \in T$ et $(X,Y) \notin T$. Commençons par étudier le cas $(X,Y) \in T$.

Cas 1. $(X,Y) \in T$

Procédure Insérer (u,v) si $(X,Y) \in T$.

1. Rechercher l'ensemble des nœuds $X,Y \in T$ tel que $u \in B_x$, $v \in B_y$. Si $(X,Y) \in T$ aller à l'étape 2 ;
2. Remplacer l'arête (X,Y) dans T par un nouveau nœud Z représentant $B_z = I \cup \{u,v\}$ et ajouter les arêtes (X,Z) et (Z,Y).

La procédure Insérer (u,v) met à jour T, telle que T' est une décomposition arborescente pour $G + (u,v)$. De plus, T' vérifie la condition des sous arbres induits pour les deux extrémités de l'arête (u,v) appartenant à $Z \in T'$, qui satisfait aux trois conditions de la définition 1.9.1.

Et on a le résultat suivant concernant la largeur et la longueur arborescente de $G + (u,v)$.

Proposition 4.4.9 *On a* $tw(G') \leq tw(G) + 1$ *and* $tl(G') = tl(G)$.

Preuve

Puisque $width(T) = max_{X \in V(T)} |X| - 1$, *si* B_x *est l'unique nœud qui détermine la largeur arborescente de* T, *alors* $tw(G') = tw(G) + 1$, *sinon* $tw(G) = tw(G')$. *Étant donné* $length(T) = max_{X \in V(T)} diam_G(X)$, *il apparaît évident que* $tl(G') = tl(G)$.

□

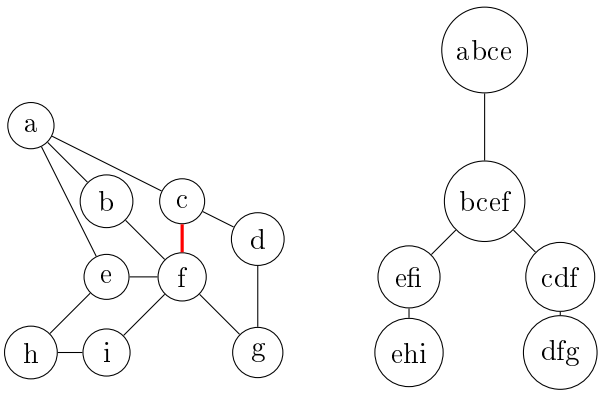


FIGURE 4.1 – c et f sont dans un même sac donc $T' = T$. De plus on a : $tw(G) = tw(G')$ et $tl(G') = 2 < tl(G) = 3$

Cas 2 $(X, Y) \notin T$.

Nous allons explicitement utiliser le fait que que si T_u et T_v sont des sous arbres de T induits par $B_G(u)$ et $B_G(v)$ respectivement, alors T_u et T_v ne s'intersectent pas (car aucun sac ne contient les sommets u et v).

Soit $T_{\overline{uv}}$ un $B_G(u)$,$B_G(v)$-séparateur minimal, alors $T_{\overline{uv}}$ est un nœud séparateur minimal dans T. De plus, $T_{\overline{uv}}$ est un sous arbre de T induit par les nœuds du séparateur minimal.

Procedure Insérer (u, v) si $(X, Y) \notin T$.

1. Rechercher l'ensemble des nœuds $X, Y \in T$ tel que $u \in B_x$, $v \in B_y$. Trouver T_u et T_v les sous arbres de T respectivement induit par $B_G(u)$ et $B_G(v)$. Et trouver $T_{\overline{uv}}$ le séparateur minimal de X et Y dans T ;
2. Comparer entre $|T_u|$ et $|T_v|$. Si $|T_u| \leq |T_v|$; alors ajouter le sommet v à chaque sac de $T_{\overline{uv}}$ et B_x et sinon, ajouter le sommet u à chaque sac de $T_{\overline{uv}}$ et B_y.

Finalement, Insérer (u, v) met à jour T et produit T' une décomposition arborescente pour $G + (u, v)$, puisque T' vérifie la propriété des sous arbres induits et les condition de la definition 1.9.1.

Proposition 4.4.10 *Soit* T' *la décomposition arborescente de* $G' = G + (u, v)$ *avec* $(X, Y) \notin T$, *alors on a ;* $tw(G') \leq tw(G) + 1$ *et* $tl(G') \leq tl(G)$.

Preuve

Si T_u *contient au moins un sac avec* $(tw(G) + 1)$ *sommets, alors l'insertion d'un sommet* v *aux sacs de* T_u *implique que* T_u *va contenir au moins un sac avec* $(tw(G) + 2)$ *sommets*

et ainsi $tw(G') = tw(G) + 1$, *Sinon on aura,* $tw(G') = tw(G)$. *Même raisonnement pour* T_v. $tl(G') \leq tl(G)$ *; il est clair que l'insertion d'une arête réduit les distances dans le graphe* G' *et par conséquent la longueur de* T', *la borne sur la longueur arborescente de* G' *est la même que pour* G.

□

Ci-dessus un exemple pour illustrer ce cas.

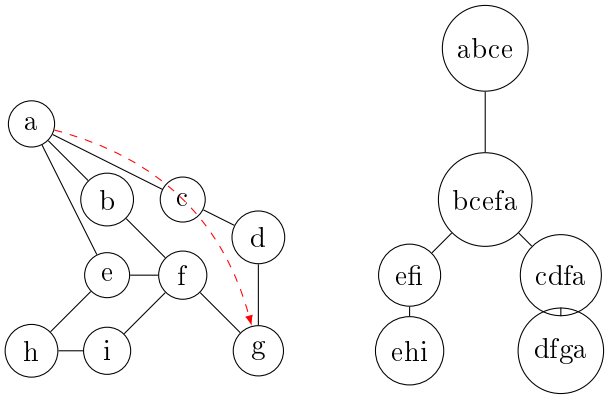


FIGURE 4.2 – On a : $tw(G') = tw(G) + 1 = 3 + 1 = 4$ et $tl(G') = tl(G) = 3$

ii)-Suppression d'une arête

Soit (u, v) l'arête à supprimer et $G' = G-(u, v)$. Nous montrons maintenant comment obtenir T' la décomposition arborescente de G'.

Partitionner l'ensemble $N(X)$ des voisins de X en les ensembles suivants :
$N_u = \{Y \in N(X) \ / \ u \in B_y\}$
$N_v = \{Z \in N(X) \ / \ v \in B_z\}$
$N_{\overline{uv}} = \{W \in N(X) \ / \ u, v \notin B_w\}$
Pour chaque nœud X de T tel que $(u, v) \in B_x$, soit $B_x^u = B_x - \{v\}$, $B_x^v = B_x - \{u\}$. Dans $G - (u, v)$, chaque sac B_x est décomposé en deux sacs B_x^u et B_x^v.
Ensuite remplacer le nœud X avec les nouveaux nœuds X_1 et X_2 respectivement représentant B_x^u et B_x^v et ajouter l'arête (X_1, X_2)
Si $Y \in N_u$, remplacer (X, Y) par (X_1, Y)
Si $Z \in N_v$, remplacer (X, Z) par (X_2, Z)
Si $W \in N_{\overline{uv}}$, remplacer (X, W) par (X_1, W) ou (X_2, W), choisi de manière arbitraire.
On observe que T vérifie la propriété des sous arbres induits. Ainsi on a la proposition suivante ;

Proposition 4.4.11 $tw(G') \leq tw(G)$ *et* $tl(G') \geq tl(G)$.

Preuve

On a $width(T) = max_{X \in V(T)} |X| - 1$, *si* B_x *est l'unique nœud qui détermine la largeur arborescente de* T, *alors, le fait que l'on décompose* B_x *en deux sacs* B_x^u *et* B_x^v, *réduit la largeur de* T, *et on a* $tw(G') < tw(G)$. *Sinon* $tw(G') = tw(G)$. *Par ailleurs, on a* $tl(G') \geq tl(G)$ *; en effet, il est clair que la suppression d'une arête dans le graphe augmente les distances dans le graphe* G' *et par conséquent la longueur de* T'.

□

Remarque 89 *1. Si $\exists B_{y_i}$ tel que $B_x^u \subset B_{y_i}$, pour quelques $Y_i \in N_u$, choisir un tel Y_i arbitrairement, contracter (X_1, Y_i) et remplacer X_1 avec Y_i. Cette opération maintient la propriété des sous arbres induits. On fait de même pour la cas où $B_x^v \subset B_{z_i}$, pour quelques $Z_i \in N_v$.*

2. Trivialement si (u, v) est une arête déconnectante dans G, alors la suppression de (u, v) déconnecte la composante connexe de G.

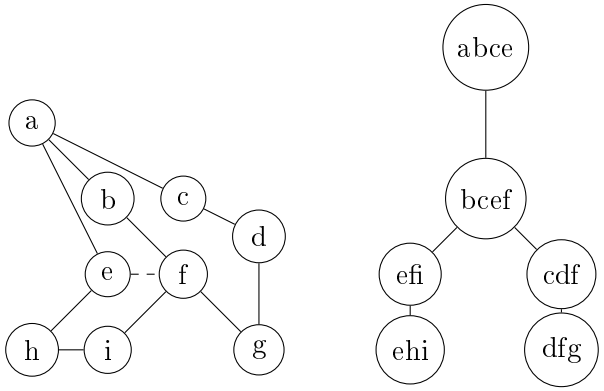


FIGURE 4.3 – On a $T' = T$. De plus on a : $tw(G') = tw(G) = 3$ et $tl(G') = tl(G) = 3$

iii)-Insertion d'un sommet

Dans cette partie, nous montrons comment mettre à jour T pour $G' = G + u$, où u est un sommet tel que $u \notin V(G)$.

Soit $E' = E(G + u)$, alors $E' = E \cup \{(u, v_i); v_i \in N_{G'(u)}$ et $v_i \in V(G)\}$.

S'il existe $X \in T$ tel que $v_i \in B_x$, $\forall i$, alors $X' = B_x \cup \{u\}$ est un sac dans T'.

Pour mettre à jour T, on remplace X par X' ; la décomposition arborescente obtenue satisfait toutes les conditions de la définition 1.9.1 pour $G' = G + u$.

Si un tel nœud n'existe pas, alors on utilisera la procédure suivante :

Procédure Insérer Sommet

1. Insérer u à chaque nœud dans T, aller à l'étape 2 ;
2. Pour chaque nœud X qui est une feuille dans T faire ; Supprimer X tant qu'il existe au moins un nœud qui contient u et v_i, $\forall i$.

À la fin de la procédure, on obtient un sous arbre noté T_u induit par $B_{G'}(u)$. Ainsi T' est obtenu à partir de T par la mise à jour des sacs du seul sous arbre T_u, les autres sacs restant inchangés.

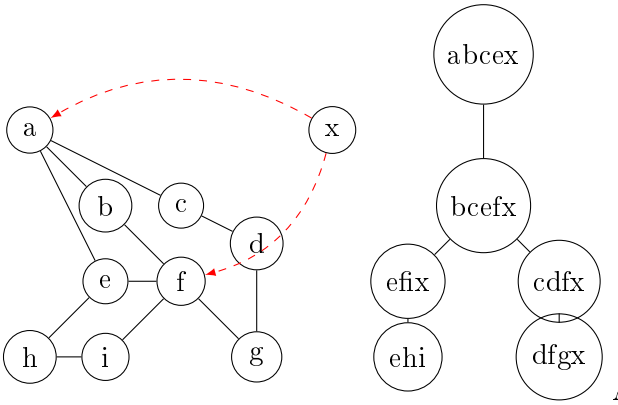
T' est une décomposition arborescente pour $G + u$ satisfaisant la propriété des sous arbres induits. Et on a le rltat suivant :

Proposition 4.4.12 *On a ; $tw(G) \leq tw(G') \leq tw(G) + 1$ et $tl(G') \leq tl(G) + 1$,*

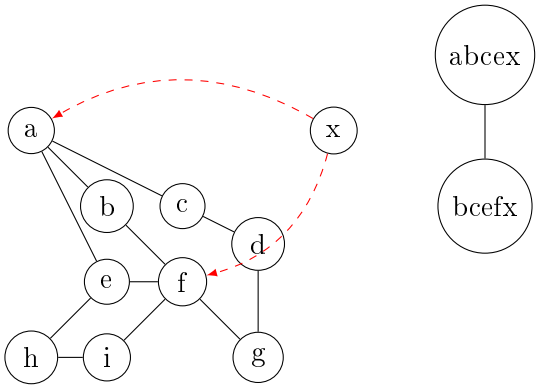
Preuve

Si B_x est l'unique nœud qui détermine la largeur arborescente de T, alors le fait que l'on ajoute un nouveau sommet à B_x va impliquer que $tw(G') = tw(G)+1$, sinon, on a $tw(G') = tw(G)$. Par ailleurs, on a : $tl(G') \leq Length(T')$, et $Length(T') = Max_{B \in V(T')} diam_{G'} B$ et $Length(T'_x) = Max_{B_x \in V(T')} diam_{G'} B_x$ avec B_x sac contenant u dans T', et $diam_{G''} B_x = max_{v,u \in B_x}\{dist_{G'}(v,u)\} \leq max_{v,u \in B_x}\{dist_G(v,u)\} + 1 = (diam_G B_x) + 1$ qui implique que $tl(G') \leq tl(G) + 1$.

□



Ajoût du sommet x



Le sous arbre T'_x

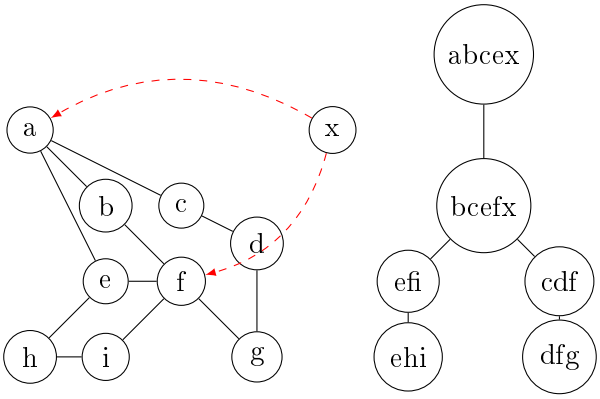


FIGURE 4.4 – La décomposition arborescente T' de G'. On a : $tw(G') = 4 > tw(G) = 3$ et $tl(G') = tl(G) = 3$

iv)-Suppression d'un sommet

Nous montrons comment mettre à jour T pour $G' = G - u$, i.e., quand on supprime un sommet u du graphe G.

Procédure

1. Pour chaque nœud $X \in T$ tel que $u \in B_x$, soit $B_{x'} = B_x - \{u\}$. Dans $G - \{u\}$, chaque sac B_x est remplacé par $B_{x'}$.

2. S'il existe B_{y_i} tel que $B_{x'} \subset B_{y_i}$, pour quelques $Y_i \in N(X')$, choisir un tel Y_i arbitrairement, contracter $\{X', Y_i\}$ et remplacer X' avec Y_i. La décomposition arborescente obtenue sera inévitablement réduite.

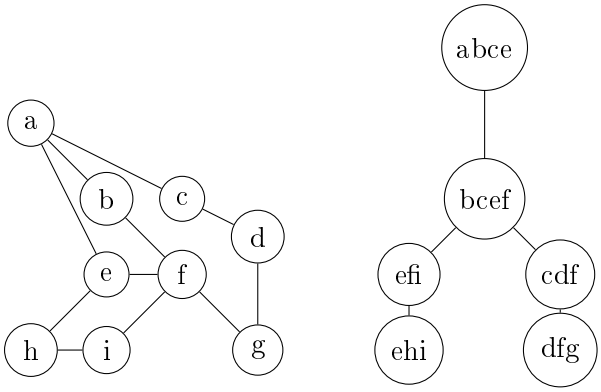
Remarque. u est un point d'articulation, quand la suppression de u déconnecte une composante connexe de G. Et l'on a :

Proposition 4.4.13 *On a ;* $tw(G) - 1 \leq tw(G') \leq tw(G)$ *et* $tl(G') \leq tl(G)$

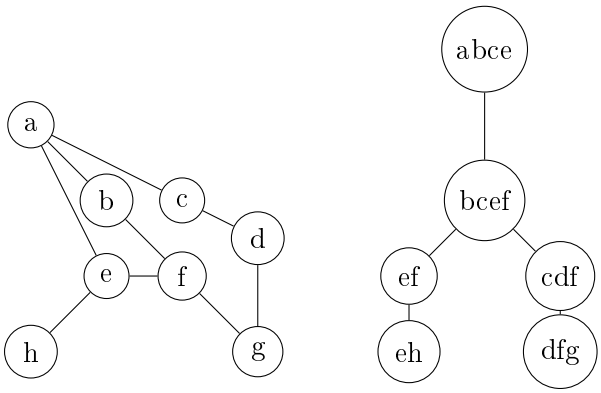
Preuve

En effet, si tous les sacs de T *qui contiennent* $(tw(G) + 1)$ *sommets sont parmi les sacs auxquels, on supprime le sommet* u*, alors ;* $tw(G') = tw(G) - 1$*. Sinon* $tw(G') = tw(G)$*. Il est évident que,* $tl(G') \leq tl(G)$ *(par définition).*

□



On a le graphe G et sa décomposition arborescente T.



Suppression du sommet i du graphe G et la décomposition arborescente T.

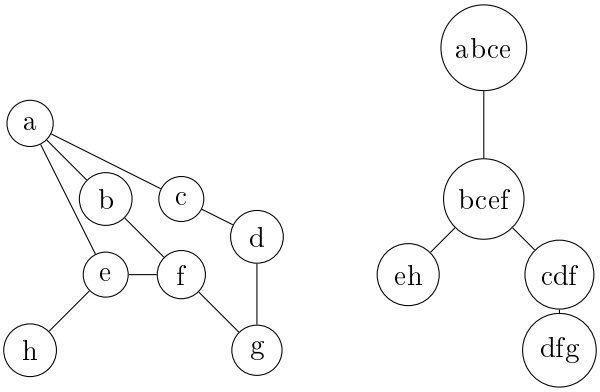


FIGURE 4.5 – Le graphe G' et sa décomposition arborescente réduite T'

4.4.6 Implémentation

A- Structure de données

Étant donné un graphe quelconque G en entrée, représenté par sa liste d'adjacence, comme étape initiale de pré-traitement, on peut trouver les degrés de tous les sommets en temps linéaire et stocker les valeurs dans une liste. Puisque l'on suppose que la représentation décomposition arborescente du graphe est donnée, les sommets peuvent être partitionnés en sacs X_i selon la

décomposition arborescente T en temps linéaire. Chaque élément dans la liste des degrés, indique son sac correspondant et le sommet dans le graphe.

De cette manière, trouver à quel ensemble appartient un sommet ou son degré correspondant, supprimer un sommet donné d'un ensemble, et ajouter un sommet donné à un ensemble se fait en temps constant. Par conséquence, le déplacement d'un ensemble de sommet donné A, se fait en temps $O(|A|)$.

Étant donné cette structure, scanner ou retourner un sommet dans la décomposition arborescente se fait en temps linéaire en la taille du nœud, puisque les listes des degrés sont toutes connectées par des pointeurs , donc on peut juste les scanner l'un après l'autre, en commençant par celui correspondant au degré minimum dans le vecteur.

Comme noté précédemment, la taille de la liste des degrés est n, mais puique nous permettons des ajouts de sommets au graphe, pour chaque opération nous devrions augmenter la taille de la liste par 1. Cette opération peut être amortie en temps constant en utilisant les listes dynamiques, ou l'allocation dès le début de la liste de taille $n+n\prime$, où $n\prime$ est une borne supérieure au nombre des sommets ajoutés possibles donné par l'utilisateur de l'algorithme. Notons que ceci est quelque chose dont tout algorithme dynamique doit faire face indépendemment de la décomposition arborescente.

B- Opérations

Dans cette partie, nous montrons comment notre structure de donnée peut être mise à jour efficacement après une modification qui maintien la représentation décomposition arborescente du graphe en entrée en un temps raisonnable quand on ajoute ou supprime une arête ou un sommet.

Avant de commencer l'implémentation, nous avons besoin de connaître la complexité de la détermination des $a, b-$ séparateurs minimaux dans les graphes qui est donné dans le papier de (Kloks et Kratsch 1998) [95]. Cet algorithme est en temps polynomial par séparateur. Plus précisément, si on note $R(a, b)$ le nombre des $a, b-$ séparateurs (pour les sommets non-adjacents a et b). L'algorithme détermine tous les $a, b-$séparateurs en $O(n^3 R(a, b))$.

Rappelons cependant que l'on suppose que la représentation décomposition arborescente T du graphe G existe.

Nous implémentons la procédure d'Insertion d'une arête, par la recherche des nœuds les plus proches X et Y tels que $u \in B_X$ et $v \in B_Y$, et déterminer si $(X, Y) \in T$ ou non. Ceci se fait en temps polynomial. On a deux cas : Si $(X, Y) \notin T$, on calcule T_u et T_v les sous arbres de T, et le $X, Y-$séparateur minimal dans T, qui se fait en temps polynomial, et si $(X, Y) \in T$, on remplace l'arête (X, Y) par le nœud Z et on ajoute les arêtes (X, Z) et (Z, Y). Puisque T possède au plus n nœuds et chaque vecteur est de longueur n, la procédure d'insertion d'une arête se fait en temps $O(n^3 R)$, où R est le nombre des $X, Y-$séparateurs minimaux.

On implémente la procédure de suppression d'une arête en recherchant un nœud X et en examinant ses voisins en temps polynomial.

On implémente la procédure d'insertion d'un sommet par la recherche des nœuds X, et l'ajout de u à chacun d'entre eux, ensuite les supprimer jusqu'à ce qu'il existe au moins un nœud qui contient u et v_i , où $v_i \in E'$. cette opération se fait en temps polynomial.

On implémente la procédure de suppression d'un sommet par la recherche des nœuds X, et de supprimer u de chacun d'entre eux, se fait en temps polynomial.

Finalement, le coût total de l'algorithme dynamique est $O(n^3R)$, où R est le nombre des $X, Y-$séparateurs minimaux.

Conclusion générale et perspectives

Dans cette partie, nous allons rappeler succinctement l'ensemble des résultats obtenus, et ensuite présenter les perspectives de recherches ouvertes dans cette thèse.

Synthèse du travail effectué

L'objet principal de cette thèse est l'étude de la décomposition arborescente des graphes, et l'application principale est le routage dans un réseau de communications qui est décrit par un graphe.

De nombreux articles de la littérature ont montré qu'il est possible de réduire significativement la taille des tables de routage des réseaux (on parle alors de routage compact) lorsque le graphe sous-jacent se décompose de manière spécifique : selon une décomposition-arborescente comme définie par Robertson et Seymour [116].

L'intérêt de telles décompositions dépasse largement le cadre du routage compact et elles sont massivement présente en algorithmique de graphes. La nature des résultats de la thèse est à la fois combinatoire (chapitre 3) et algorithmique (chapitre 4). Les quatres chapitres de la thèse sont les suivants :

Dans le premier chapitre, nous présentons rapidement les différentes notions liées à la théorie des graphes et à la décomposition arborescente. Nous avons notamment présenté les notions de "largeur arborescente" et de "longueur arborescente" qui sont des mesures d'efficacités de ces décompositions. La largeur arborescente est la mesure classique et originale introduite par Robsertson et Seymour [116], alors que la longueur (liée à la distance entre deux sommets d'un même sac de la décomposition) a été introduite une vingtaine d'années plus tard par Dourisbourre et Gavoille [52].

Dans le chapitre 2, nous présentons un état de l'art sur l'application visée, à savoir le routage compact. Les tenants et les aboutissants sont expliqués ainsi que les résultats principaux de la littérature qui montrent que les graphes possédant une décomposition arborescente de longueur bornée, possèdent un schéma de routage compact presque de plus courts chemins (déviation constante par rapport aux plus courts chemins) avec des table et des adresses de taille poly-logarithmique en n (n étant le nombre de sommets du graphe).

Dans le chapitre 3 nous présentons notre première contribution de la thèse : les graphes k-cordaux, c'est-à-dire sans cycle de longueur $> k$ sont de longueur arborescente au plus $\lceil k/3 \rceil$. Ce résultat a été publié dans une conférence [22] et soumis à un journal [29]. Il généralise le résultat de Dourisbourre et al. établit pour les graphes planaires-extérieurs.

Le chapitre 4, constitue le noyau central des contributions de cette thèse. On y présente des algorithmes dynamiques pour maintenir une décomposition arborescente supposée donnée au départ. Plus précisément, l'algorithme permet de maintenir la longueur ou la largeur arborescente pour les graphes faiblement triangulés et même les graphes arbitraires, avec la suppression ou

l'ajout de sommets ou d'arêtes. Le résultat était connu pour les graphes triangulés, et il est étendu pour celui des faiblement triangulés avec un coût de $O(m^2)$ par opération où m est le nombre d'arête du graphe. Pour le cas général, le coût de chaque opération s'exprime par un polynôme en le nombre de séparateurs minimaux entre deux sommets du graphe. Cette partie a fait l'objet de plusieurs publications et soumissions.

Nous proposons de conclure cete thèse en précisant les pistes pour guider d'éventuels travaux futurs sur l'étude des paramètres de graphes et leur maintien par des algorithmes dynamiques et éventuellement pour l'amélioration des résultats obtenus.

Perspectives

Dans cette partie, nous allons poser l'ensemble des points ouverts dans cette thèse, et qui feront l'objet de réflexions dans un futur proche.

Une extension possible aux travaux présentés dans notre thèse consiste à étendre les résultats présentés au quatrième chapitre à la recherche d'algorithmes dynamiques pour maintenir une décomposition arborescente dans le cas d'ajout et de suppression d'une clique K. L'algorithme pourra aussi permettre de maintenir la longueur ou la largeur arborescente pour les graphes faiblement triangulés et plus gnéralement k-cordaux ($k \geq 5$) dans les cas de la suppression ou l'ajoût d'une clique K de cardinalité supérieure ou égale à 3. Un résultat étant connu pour les graphes triangulés [101]. Il serait intéressant de le généraliser pour les graphes faiblement triangulés avec une complexité polynomial.

Un deuxième point consiste en l'amélioration de la complexité des algorithmes dynamiques présentés ici. On pourra utiliser les structures de données plus adaptées aux classes de graphes considérées et tirer avantage de leurs structures respectives.

Un troisième point est de considérer d'autres types de décompositions de graphes comme le clustering des graphes par exemple. Cette décomposition de graphes a beaucoup d'applications dans les domaines des réseaux de télécommunication, des problèmes liés à la classification, data mining et le traitement d'images. Les algorithmes dynamiques prendront en charge la mise à jour de certains paramètres liés à chaque domaine d'application.

En dernier, il serait important de réfléchir à la conception d'algorithmes de routage dynamique pour le problème du routage compact. Ces algorithmes seraient adaptés à prendre en charge les pannes survenues au réseau, pannes consistant en la suppression de sommets et/ou d'arêtes. Ces algorithmes doivent être capables de rerouter les messages dans chacun des cas avec de nouvelles adresses et mémoires locales de taille pseudo-logarithmiques et des fonctions de routage calculables en temps logarithmique.

Bibliographie

[1] 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.

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

[3] S. Arnborg and A. Prokurowski. *A linear time algorithm for NP-Hard problems restricted to partial k-trees.* Discrete Applied Mathematics, 23 (1989), pp.11-24.

[4] S. Arnborg. *Efficient algorithms for combinatorial problems on graphs with bounded decomposability, A survey.* BIT, 25 (1985), pp.2-23.

[5] P.A. Bernstein and N. Goodman. *Power of natural semi joins.* Siam Journal on Computing 10(4) 751-771, 1981.

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

[7] A. Berry, J-P. Bordat and P. Heggernes. *Recognizing Weakly Triangulated Graphs by Edge Separability.* Nordic Journal Computing, 7(3) (2000) pp.164 - 177.

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

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

[10] A. Berry, J-P. Bordat, P. Heggernes, G. Simonet and Y. Villanger. *A wide-range algorithm for minimal triangulation from an arbitrary ordering.* Journal of Algorithms, Volume 58, Issue 1,(2006), pp. 33 - 66.

[11] A. Berry. *Désarticulation d'un graphe.* PhD thesis, Université Montpellier II, 1998.

[12] A. Berry. *A Wide-Range Efficient Algorithm for Minimal Triangulation.* In Proceedings of SODA'99 SIAM Conference, january 1999.

[13] H. Bodlaender, T. Kloks and D. Kratsch. *Treewidth and pathwidth of permutation graphs.* SIAM J. Discrete Math.,8(4) 606-616, 1995.

[14] H. Bodlaender, A. Koster, F. van den Eijkhof and L. van der Gaag. *Prepocessing for triangulation of probabilistic networks.* Uncertainty in Artificial Intelligence, pages 32-39, (2001).

[15] H.L. Bodlaender. *Dynamic programming on graphs of bounded treewidth.* Lecture Notes Computer Sciences (317) page 105-119, Springer-verlag, 1988.

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

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

[18] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University press, 1998.

[19] V. Bouchitté and I. Todinca. *tree-width and minimum fill-in.* grouping the minimel separators. SIAM, J on Computing, 31 (1) : 212 - 232, 2001.

[20] V. Bouchitté and I. Todinca. *Listing all potential maximal cliques of a graph.* Theoret. Comput. Sci., 276(1-2) :17-32, 2002.

[21] **M.A. Boutiche**. *Graphes et problèmes de routage.* Master's Thesis, Université des Sciences et de la Technologie d'Alger 2005.

[22] **M.A. Boutiche**, H. Ait Haddadène and H.A. Le Thi. *Tree-length of some graph classes and application to the routing problem.* Actes du Colloque sur l'Optimisation et les Systèmes d'Information COSI'06, Annales ROAD Research Report 06, (2006), pp.1-12.

[23] **M.A. Boutiche**. *Control of Some Graph Invariants in Dynamic Routing.* In Proceeding of the Second International Conference MCO'08, SpringerBook CCIS Volume 14, pp. 52-58 (2008).

[24] **M.A. Boutiche**. *Reliabity of Tree Decomposition Graph Model Networks.* Actes du Colloque sur l'Optimisation et les Systèmes d'Information COSI'08, Tizi-Ouzou (2008).

[25] **M.A. Boutiche**. *Control of Some Graph Invariants.* In Proceeding of the International Symposium on Operational Research (ISOR'08), Annales ROAD, pp.181-188 (2008).

[26] **M.A. Boutiche**. *Topology Control and Maintaining of Graph Invariants.* Studia Informatica Universalis, 9(2)(2011) 38 - 49.

[27] **M.A. Boutiche**. *A dynamic algorithm for maintaining some graph invariants for weakly chordal graphs.* Confnce ROADEF 2011, Du 02 au 04 Mars 2011 Saint-Etienne.

[28] **M.A. Boutiche**, H. Ait Haddadène and H.A. Le Thi. *Maintaining Graph Properties of Weakly Chordal Graphs.* Applied Mathematical Sciences, Vol. 6, no. 16, 765 - 778, 2012.

[29] **M.A. Boutiche**, H. Ait Haddadène and H.A. Le Thi. *Tree-length of classes of graphs.* submitted to Utilitas Mathematica, (2012).

[30] **M.A. Boutiche**. *Maintenance des graphes faiblement triangulés.* Rencontre d'Analyse Mathématiques et ses Applications (RAMA8), 26 - 29 Nov, Alger (2012).

[31] H.J. Broersma, E. Dahlhaus and T. Kloks. *Algorithms for the treewidth and minimum fill-in of HHD-free graphs.* in Workshop on graphs (WG'97) volume 1335 of Lecture Notes in Computer Science, page 109-117, Springer-verlag, 1997 .

[32] H.J. Broersma, E. Dahlhaus and T. Kloks. *A linear time algorithms for minimum fill-in and treewidth for distance-hereditary graphs.* in Scientific program 5th Twente Workshop on graphs and Combinatorial Optimization, pages 48-50, 1997.

[33] P. Buneman. *A characterisation of rigid circuit graphs.* Discrete Mathematics, 9 (1994), pp. 205-212.

[34] S.R. Buss. *Toward NP-P via proof complexity and search.* Annals of Pure and Applied Logic, 163 : 906-917, 2012.

[35] K. Cameron, R. Sritharan and Y. Tang. *Finding a maximum induced matching in weakly chordal graphs.* Discrete Mathematics, 266 : 133 - 142, 2003.

[36] 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 Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, May 2003, pp. 96-107.

[37] M.S. Chang. *Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs.* in (ISAAC'96) volume 1178 of Lecture Notes in Computer Science, page 146-155, Springer-verlag, 1996.

[38] T. Cormen, C. Leiserson and R. Rivest. *Introduction algorithmique.* Dunod, 1994.

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

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

[41] B. Courcelle. *Décompositions Arborescentes.* Dans Graphes et applications 2, Jean-Claude Fournier (Ed.) (2007), pp. 195-231.

[42] C. Crespelle and C. Paul. *Fully dynamic algorithm for recognition and modular decomposition of permutation graphs.* In 31th Int. Workshop on Graph Theoretical Concepts in Computer Science (WG05), volume 3787 of Lecture Notes in Computer Science, page 38-48, Springer-verlag, 2005.

[43] C. Crespelle and C. Paul. *Fully dynamic recognition algorithm and certificate for directed cographs.* Discrete Applied Mathematics, 154 (2006), pp. 1722-1741.

[44] C. Crespelle. *Représentations dynamiques de graphes.* PhD Thesis University of Montpellier II, 2007.

[45] G. Di Battista and R. Tamassia. *On-line planarity testing.* SIAM Journal on Computing, 25 (1996), pp. 956-997.

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

[47] Y. Dieng. *Décomposition arborescente des graphes planaires et routage compact.* PhD Thesis, Université Bordeaux I, 2009.

[48] G.A. Dirac. *On rigid circuit graphs.* Abh. Math. Sem. Univ. Hamburg, 21 : 71-76, 1961.

[49] Y. Dourisboure and C. Gavoille. *Improved compact routing scheme for chordal graphs.* in 16th International Symposium on DIStributed Computing (DISC), Lecture Notes in Computer Science, vol. 2508, Springer, Berlin, 2002, pp. 252 - 264.

[50] Y. Dourisboure. *Routage compact et longueur arborescente.* PhD Thesis LaBRI, University of Bordeaux I 2003.

[51] Y. Dourisboure. *Compact routing schemes for generalised chordal graphs.* Journal of Graph Algorithms and Applications, volume 9, number 2, pages 277-297, (2005).

[52] Y. Dourisboure and C. Gavoille. *Tree decomposition with bags of small diameter.* Discrete Mathematics, Vol 307, pp. 2008-2029 2007.

[53] F.F. Dragan and I. Lomonosov. *On compact and efficient routing in certain graph classes.* Discrete Applied Mathematics, Vol 155, pp.1458-1470, 2007.

[54] D. Eppstein, Z. Galil and G.F. Italiano. *Dynamic graph algorithms.* In MJAtallah, editor, Algorithms and theory of computation handbook, chapter8. CRC Press, Boca Raton, FL, 1998.

[55] D. Eppstein, Z. Galil, G.F. Italiano and T.H. Spencer. *Separator based sparsification II : edge and vertex connectivity.* SIAM J. on Computing.,28(1) 341-381, 1998.

[56] J. Feigenbaum and S. Kannan. *Dynamic graph algorithms.* In Rosen Editor, Handbook of discrete and combinatorial mathematics, chapter 17. CRC Press, Boca Raton, FL, 1999.

[57] A. Fiat and G.J. Woeginger. *Online algorithms : The state of the art.* Number 1442 in LNCS State of the art survey. Springer, 1998.

[58] P. Fraignaud and C. Gavoille. *Routing in trees.* In 28th international colloquium on automata, languages and programming (ICALP), volume 2076 LNCS, pages 757-772 (2001).

[59] M.L. Fredman and M.E. Saks. *he cell probe complexity of dynamic data structures.* In STOC., 345-354, 1989.

[60] T. Fujisawa and H. Orino. *An efficient algorithm of finding a minimal triangulation of a graph.* In IEEE International Symposium on Circuits and Systems, pages 172-175, 1974.

[61] D.R. Fulkerson and O.A. Gross. *Incidence matrixes and interval graphs.* Pacific Journal of Math., 15 : 835-855, 1965.

[62] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W. H. Freeman and Co (1979).

[63] C. Gavoille. *Structures de données compactes et disctribuées.* HDR, Labri de Bordeaux, 2000.

[64] C. Gavoille, M. Katz, N.A. Katz, C. Paul and D. Peleg. *Approximate distance labeling schemes.* in F.M. auf der Heide (Ed.), Ninth Annual European Symposium on Algorithms (ESA), Lecture Notes in Computer Science, vol. 2161, Springer, Berlin, 2001, pp.476-488.

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

[67] V. Gogate and R. Dechter. *A complete anytime algorithm for treewidth.* In Proceedings of UAI, pages 201-208, (2004).

[68] M.C. Golumbic, M. Lypshteyn and M. Stern. *Intersection Models of Weakly Chordal Graphs.* Discrete Applied Math, 157 (9) (2009) 2031 - 2047.

[69] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[70] R. Grone, C.R. Johnson, E.M. Sá and H. Wolkowicz. *Positive definite completions of partial Hermitian matrices.* Linear Algebra and Its Applications., 58 : 109-124, 1984.

[71] J. Håstad. *Clique is hard to approximate within* $n^{1-\epsilon}$. Acta Math.,182(1) 105-142, 1999.

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

[73] T. Hagerup. *Dynamic Algorithms graphs of bounded treewidth.* in (ICALP'97) Lecture Notes in Computer Science, page 292-302, Springer-verlag, 1997.

[74] P.L. Hammer and B. Simeone. *The splittance of a graph.* Combinatorica Vol 1 (3), pp. 275-284 (1981).

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

[76] R. Hayward. *Generating weakly triangulated Graphs.* Journal of Graph Theory, 21, 67 - 70, 1996.

[77] R.B. Hayward, C.T. Hoang, and F. Maffray. *Optimizing weakly triangulated graphs.* Graphs and Combinatorics 5(1) :339-349, 1989.

[78] R. Hayward, J. Spinrad and R. Sritharan. *Weakly Chordal Graph algorithms via handles.* In Proceeding of Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2000.

[79] P. Heggernes. *Minimal triangulations of graphs : A survey.* Discrete Mathematics., (306) 3 297-317, 2006.

[80] P. Heggernes and F. Mancini. *Dynamically maintaining split graphs.* Discrete Applied Mathematics., (157) 9 2057-2069, 2009.

[81] P. Hell, R. Shamir and R. Sharan. *A fully dynamic algorithm for recognizing and representing proper interval graphs.* SIAM J Computing, 31 : 289 - 305, 2001.

[82] P. Hell, R. Shamir, and R. Sharan. *A fully dynamic algorithm for recognizing and representing proper interval graphs.* SIAM Journal on Computing.,31(1) 289-305, 2002.

[83] M.R. Henzinger and M.L. Fredman. *Lower bounds for fully dynamic connectivity problems in graphs.* Algorithmica.,22(3) 351-362, 1998.

[84] 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.

[85] J. Holm, K. de Lichtenberg, and M. Thorup. *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity.* In STOC., 79-89, 1998.

[86] W-L. Hsu. *On-line recognition of interval graphs in $o(m + n \log n)$ time.* Combinatorics and Computer Science., 27-38, 1996.

[87] L. Ibarra. *Fully dynamic algorithms for chordal graphs.* In 10th ACM-SIAM Annual Symposium on Discrete Algorithm SODA99., 923-924, 1999.

[88] L. Ibarra. *A fully dynamic algorithm for recognizing interval graphs using the clique-separator graph.* Technical Report DCS-263-IR, Dept. of Computer Science, University of Victoria, 2001.

[89] L. Ibarra. *Fully dynamic algorithms for chordal graphs and split graphs.* ACM Transactions on Algorithms, Vol 4 n4, pp. 1-20 (2008).

[90] L. Ibarra. *A Fully dynamic graph algorithm for recognizing proper interval graph.* WALCOM'09, pp. 190-201 (2009).

[91] L. Ibarra. *A Fully dynamic graph algorithm for recognizing interval graphs.* Algorithmica, Vol 58 n3, pp. 637-678 (2010).

[92] U. Kjaerulff. *Triangulation of Graphs - Algorithms Giving Small Total State Space.* Technical report, Judex R.R. Aalborg, Denmark, 1990.

[93] U. Kjaerulff. *Triangulation of graphs - algorithms giving small total state space.* Technical report, Judex R.R. Aalborg, Denmark, 1990.

[94] T. Kloks and D. Kratsch. *Treewidth of chordal bipartite graphs.* J. Algorithms.,19(2) 266-281, 1995.

[95] T. Kloks and D. Kratsch. *Listing all Minimal Separators of a Graph.* SIAM Journal on Computing, Vol 27, Issue 3, pp. 605 - 613 (1998).

[96] T. Kloks, D. Kratsch, and H. Müller. *Approximating the bandwidth for asteroidal triple-free graphs.* In in 3-rd Annual European Symposium on Algorithms (ESA95), volume 979 LNCS, pages 434-447 (1995).

[97] T. Kloks, D. Kratsch and J. Spinrad. *Treewidth and pathwidth of cocomparability graphs of bounded dimension.* Res. Rep. 93-46, Eindhoven University of Technology, 1993.

[98] T. Kloks, D. Kratsch, and J. Spinrad. *On tree-width and minimum fill-in of asteroidal triple-free graphs.* Theoretical Computer Science, 175 (1997), pp.309 - 335.

[99] T. Kloks, D. Kratsch and C.K. Wong. *Minimum fill-in of circle and circular-arc graphs.* J. Algorithms.,28(2) 272-289, 1998.

[100] T. Kloks. *Treewidth of circle graphs.* Inter. J. of Found. Comp. Sci.,7 : 111-120, 1996.

[101] Y.T. Kyzy. *A Fully Dynamic Algorithm for Recognizing and Representing Chordal Graphs.* In in 6-th International Andrei Ershov Memorial Conference, (PSI'06), volume 4378 LNCS, pages 481-486 (2006).

[102] C.G. Lekkerkerker and J.Ch. Boland. *Representation of a finite graph by a set of intervals on the real line.* Fund. Math 51 (1962), pp.45 - 64.

[103] X.Y. Li. *Topology control in wireless ad hoc networks.* Ad Hoc Networking, S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, eds., IEEE Press 2003.

[104] D. Lokshtanov. *On the Complexity of Computing Treelength.* Discrete Applied Mathematics 158(7) : 820-827 (2010).

[105] G. Lueker. *Structured breadth first search and chordal graphs.* Technical Report 158, Princeton Univ, 1974.

[106] M. Mezzini. *Fully dynamic algorithm for chordal graphs with $o(1)$ query-time and $o(n^2)$ update-time.* Theoretical Computer Science 445 : 82-92 (2012).

[107] S.N. Ndiaye. *Calcul et exploitation de recouvrements acycliques pour la résolution de (V)CSP.* PhD Thesis, Paul Cézanne University of Marseille, 2007.

[108] D. Nikolopoulos, L. Palios, and C. Papadopoulos. *A fully dynamic algorithm for the recognition of P_4-sparse graphs.* In 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG06), Lecture Notes in Computer Science, 2006.

[109] D. Nikolopoulos, L. Palios, and C. Papadopoulos. *A fully dynamic algorithm for the recognition of P_4-sparse graphs.* Theoretical Computer Science 439 : 41-57 (2012).

[111] T. Ohtsuki, L.K. Cheung and T. Fujisawa. *Minimal triangulation of a graph and optimal pivoting ordering in a sparse matrix.* J. Math. Anal. Appl, 54 : 622-633, 1976.

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

[112] A. Parra and P. Scheffler. *Characterizations and algorithmic applications of chordal graph embeddings.* Discrete Applied Math., 79(1-3) :171188, 1997

[113] V.T. Paschos. *Complexité et approximation polynomiale.* Hermès-Lavoisier (2004).

[114] D. Peleg. *Proximity-preserving labeling schemes and their applications.* In 25th International Workshop, Graph-Theoretic concepts in computer science (WG), volume 1665 of LNCS Springer, pages 30-41 (1999).

[115] R. Rajaraman. *Topology control and routing in ad hoc networks : A survey.* SIGACT News, 33 pp. 60-73 (2002).

[116] N. Robertson and P.D. Seymour. *Graph minors : Algorithmic aspects of tree-width.* Journal of Algorithms, Vol 7, pp. 309-322 1986.

[117] N. Robertson and P.D. SeymouR. *Graph minors XIII The disjoint paths problem.* Journal of Combinatorial Theory Series B, Vol 63(1), pp. 65-110 1995.

[118] D. Rose, R. Tarjan, and G. Lueker, *Algorithmic Aspects of Vertex Elimination on Graphs.* SIAM Journal on computing, 5 :266283, 1976.

[119] P. Scheffler. *Linear-time algorithms for NP-complete problems restricted to partial k-trees.* Tech. Rep. R-Math-87/03, Akademie der Wissenschaften der DDR, Karl-Weierstrass-Institut fr Mathematik, 1987.

[120] R. Shamir and R. Sharan. *A fully dynamic algorithm for modular decomposition and recognition of cographs.* Discrete Applied Mathematics, 136 (2004) 329 - 340.

[121] K. Shoikhet and D. Geiger. *A practical algorithm for finding optimal triangulation.* In Proceedings of AAAI, pages 185-190, 1997.

[122] L.S. Skeide. *Recognizing weakly chordal graphs.* PhD Thesis, (2002).

[123] J. Spinrad and R. Sritharan. *Algorithms for weakly triangulated graphs.* Discrete Applied Mathematics, 59 (1995) 181 - 191.

[124] S. Sundara, K. Sher Singh and C. Pandu Rangan. *Treewidth of circular-arc graphs.* SIAM J. Discrete Math.,7 : 647-655, 1994.

[125] R. 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 (3) :566579, 1984.

[126] M. Thorup and U. Zuick. *Approximate distance oracle.* In 33^{rd} annual ACM Symposium on theory of computing (STOC), Hersonisos, Greece, ACM Press pages 1-10 (2001).

[127] I. Todinca. *Aspests algorithmiques des triangulations minimales des graphes.* PhD thesis, école normale supérieure de Lyon 1999.

[128] K. Wagner. *Über eine Eigenschaft der ebenen Komplexe.* Mathematische Annalen, 114(1) (1937) 570 - 590.

[129] J. Walter. *Representations of rigid cycle graphs.* PhD thesis, Wayne State University, USA, 1972.

[130] T. V. Wimer. *Linear algorithms on k-terminal graphs.* Ph.D. thesis, Clemson University, 1987.

[131] A. Chi-Chih. Yao. *Should tables be sorted.* Journal of ACM, 28(3) (1981) 615 - 628.

[132] D. Zuckerman. *Linear degree extractors and the inapproximability of Max clique and chromatic number.* Theory Computing, 3 (2007) 103 - 128.

Résumé

L'objet principal de cette thèse est l'étude de la décomposition arborescente des graphes. Notre principale motivation a été le routage dans un réseau de communications qui est décrit par un graphe.

De nombreux articles de la littérature ont montré qu'il est possible de réduire significativement la taille des tables de routage des réseaux (on parle alors de routage compact) lorsque le graphe sous-jacent se décompose de manière spécifique : selon une décomposition-arborescente comme définie par Robertson et Seymour [116].

L'intérêt de telles décompositions dépasse largement le cadre du routage compact et elles sont massivement présente en algorithmique de graphes. La nature des résultats de la thèse est à la fois combinatoire et algorithmique.

Dans notre thèse, nous avons commençé par prnter rapidement les différentes notions liées à la théorie des graphes et à la décomposition arborescente. Nous avons notamment présenté les notions de "largeur arborescente" et de "longueur arborescente" qui sont une mesure d'efficacité de ces décompositions. La largeur arborescente est la mesure classique et originale introduite par Robsertson et Seymour [116], alors que la longueur (liée à la distance entre deux sommets d'un même sac de la décomposition) a été introduite une vingtaine d'ann plus tard par Dourisbourre et Gavoille [52].

Un état de l'art sur l'application visée, à savoir le routage compact est aussi présenté. Le tenant et les aboutissant sont expliqués ainsi que les résultats principaux de la littératures qui montrent que les graphes possédant une décomposition arborescente de longueur bornée possède un schéma de routage compact presque de plus courts chemins (déviation constante par rapport aux plus courts chemins) avec des table et des adresses de taille poly-logarithmique en n (n étant le nombre de sommets du graphe).

Notre première contribution de la thèse est que les graphes k-cordaux, c'est-à-dire sans cycle de longueur $> k$ sont de longueur arborescente au plus $\lceil k/3 \rceil$. Ce résultat a été publié dans une conférence [22] et soumis à un journal [28]. Il généralise le résultat de Dourisbourre et al. établit pour les graphes planaires-extérieurs.

Le noyau central des contributions de cette thèse constitue l'élaboration des algorithmes dynamiques pour maintenir une décomposition arborescente supposée donnée au départ. Plus précisément, l'algorithme permet de maintenir la longueur ou la largeur arborescente pour les graphes faiblement triangulés et même les graphes arbitraire, avec la suppression ou l'ajoût de sommets ou d'arêtes. Le résultat était connus pour les graphes triangulés, et il est étendu pour celui des faiblement triangulés avec un coût de $O(m^2)$ par opération où m est le nombre d'arête du graphe. Pour le cas général le coût de chaque opération s'exprime par un polynôme en le nombre de séparateur minimaux entre deux sommets du graphe.

Mots-clés: Algorithme dynamique, graphe faiblement triangulés, graphes k-cordaux, Largeur et longueur arborescente.

Abstract

The main objective of our thesis consists in the study of the tree decomposition of graph. Our motivation consists in communication network routing described by graphs.

Many papers in the literature show that is possible to reduce significatively routing table size of networks (this is called compact routing) when the underlying graph admits tree decomposition in the sense of Robertson et Seymour [116].

The interest of such decompositions goes well beyond compact routing schemes and are massively present in graph algorithmic. The nature of the results of the thesis are both combinatorial and algorithmic

In our thesis, we first briefly introduce the different concepts related to graph theory and tree decomposition. In particular, we introduced the notions of "tree-width" and "tree length" which is an efficiency measure of these decompositions. The tree width is classical and original measure introduced by Robsertson and Seymour [116], while the tree-length (related to the distance between two vertices of the same bag of the decomposition) was introduced twenty years later by Dourisbourre and Gavoille [52].

A state of the art on the intended application, namely the compact routing is also presented. Holding and leading are explained as well as the main results of the literature showing that the graphs with a tree decomposition of bounded tree-length has a compact routing scheme almost shortest paths (constant deviation from the shortest path) with Address table in poly-logarithmic size n (n is the number of vertices of the graph).

Our first contribution of this thesis is that the k-chordal graphs, that are without cycle of length $> k$, have tree-length at most $\lceil k/3 \rceil$. This result was published in conference [22] and submitted to a journal [28]. It generalizes the result of Dourisbourre et al. provided for external planar graphs.

The core contribution of this thesis is the development of dynamic algorithms for maintaining a given tree decomposition. Specifically, the algorithm maintains the tree-length or tree-width for weakly chordal graphs and even arbitrary graphs with the removal or insertion of vertices or edges. The result was known for chordal graphs, and it is extended to the weakly chordal graphs with a cost of $O(m^2)$ per operation, where m is the number of edges of the graph. For the general case the cost for each operation is expressed by a polynomial in the least number of separator between two vertices.

Keywords: Dynamic algorithm, Weakly chordal graph, k-chordal graph, Tree-width, Tree-length.